Veracode, Inc. et al v. Appthority, Inc. Doc. 51

UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS

VERACODE, INC., and ROVI SOLUTIONS CORP.,	)	
Plaintiffs,	)	CIVIL ACTION NO. 12-10487-DPW
v.	)	
APPTHORITY, INC.,	)	
Defendant.	)	

MEMORANDUM AND ORDER
October 9, 2013

Plaintiffs Veracode, Inc. and Rovi Solutions Corp. bring this action against Defendant Appthority, Inc. for infringement of two patents for use in analyzing and manipulating computer code: U.S. Patent No. 5,854,924 (the "'924 Patent"), and U.S. Patent No. 7,752,609 (the "'609 Patent"). Veracode is the exclusive licensee of the '924 Patent, a "static debugging tool . . . to detect the presence of program errors and potential errors" in the machine-code version of a piece of software without actually running the analyzed software. ('924 Patent, Abstract.) It also owns the '609 Patent, a method of decompiling machine code, which humans cannot interpret, into a form "that one of certain skill can analyze." ('609 Patent, Summary 2:56-57.) Before me are the parties' respective briefs regarding the construction of several claim terms in the patents. The parties dispute the construction of nine terms: five terms in the '924

Dockets.Justia.com

Patent and four in the ‘609 Patent. In addition to disagreement over the precise definition of certain of these terms, the parties also dispute whether the term “Data Flow Signatures” is indefinite and whether the preamble to Claim 1 of the ‘924 Patent is a substantive limitation on the claim or merely prefatory.

I. BACKGROUND

Veracode is a computer security company, founded in 2006, providing a cloud-based platform to analyze software applications for flaws and security risks, as well as remediation services to help developers fix the flaws in their code. Similarly, Veracode’s competitor, Appthority, provides a cloud-based platform, which became publicly available in 2012 and that analyzes the enterprise risk for mobile phone applications, Appthority’s platform focuses on identifying malware and risky behaviors within applications to determine whether they are safe for the target user or company.

Software developers write computer programs in the source code of a particular programming language, such as Java, PHP, or C++. Persons of ordinary skill in the art of software development can easily read and interpret this code, but computers themselves cannot. Before a computer can read and execute the code to run the program, the source code must be compiled into machine-readable, binary code. Different computers use different binary languages. Developers can translate this

binary code into a human-readable intermediate form though a process called disassembly or decompilation. The intermediate file is intelligible to humans, but more difficult to interpret than the original source code. The intermediate file contains all the same commands and instructions as the binary code, but in a human-readable form rather than binary form. This includes the control flow - the sequence of instructions in the program - as well as the data flow - how and when the program reads and writes data into memory. Developers can then reverse engineer the intermediate file to reconstruct or approximate the original source code.

A. The Patents

Veracode filed this action on March 16, 2012 - approximately one month after Appthority's public launch - alleging infringement of the two patents in suit: the '924 Patent and the '609 Patent. Both Patents relate to software analysis.

The '924 Patent, issued in 1998, is a "Static Debugging Tool." ('924 Patent, Abstract.) It analyzes the binary version of a piece of software and, in at least one embodiment, generates an intermediate file in order to analyze the program for errors and potential errors without having to actually run the program. (*Id.* at 1:64-2:1.)

The '609 Patent, issued in 2010, but claiming priority to 2002, is a "Software Analysis Framework" which also generates an

intermediate file from a program's binary code. It "determine[s] if flaws, security vulnerabilities, or general quality issues exist in the code" by analyzing the sequence of instructions in the software as well as the processes for reading and writing data, generating a symbolic model of these processes, and comparing them to reference models. ('609 Patent, Abstract.)

II. PRINCIPLES OF CLAIM CONSTRUCTION

The purpose of the claim construction process is "to determine the meaning and scope of the patent claims that the plaintiff alleges have been infringed." *Every Penny Counts, Inc.*, v. *American Express Co.*, 563 F.3d 1378, 1381 (Fed. Cir. 2009). Claim construction is an issue of law "exclusively within the province of the court." *Markman* v. *Westview Instruments, Inc.*, 517 U.S. 370, 372, 384 (1996).

A patent must include "one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention." 35 U.S.C. § 112(b). To interpret the terms in a claim, "we look to the words of the claims themselves, the specification, the prosecution history, and any relevant extrinsic evidence." *Retractable Technologies, Inc.* v. *Becton, Dickinson and Co.*, 653 F.3d 1296, 1303 (Fed. Cir. 2011). "As a general rule, claim terms should be given their ordinary and customary meaning to persons of skill in the art as of the effective date of the patent application." *Eon-Net LP* v. *Flagstar Bancorp*, 653 F.3d 1314, 1320 (Fed. Cir. 2011). "[T]he

claims themselves provide substantial guidance as to the meaning of particular claim terms." *Abbott Laboratories* v. *Sandoz, Inc.*, 566 F.3d 1282, 1288 (Fed. Cir. 2009)(quoting *Phillips v. AWH Corp.*, 415 F.3d 1303, 1314 (Fed. Cir. 2005)). "When the ordinary meaning of claim language as understood by a person of skill in the art is readily apparent even to lay judges, claim construction 'involves little more than the application of the widely accepted meaning of commonly understood words.'" *Millipore Corp.* v. *W.L. Gore & Assoc.*, 750 F. Supp. 2d 253, 264 (D. Mass. 2010) (quoting *Phillips*, 415 F.3d at 1314).

When terms are ambiguous, the court may consult the specification to clarify their meaning. *Teleflex, Inc.* v. *Ficosa N. Am. Corp.*, 299 F.3d 1313, 1325 (Fed. Cir. 2002). However, in consulting the specification, courts must walk a fine line. We "must take care not to import limitations into the claims from the specification." *Abbott Labs* v. *Sandoz, Inc.*, 566 F.3d 1282, 1288 (Fed. Cir. 2009); *see also Kara Tech., Inc.* v. *Stamps.com Inc.*, 582 F.3d 1341, 1348 (Fed. Cir. 2009) ("The patentee is entitled to the full scope of his claims, and we will not limit him to his preferred embodiment or import a limitation from the specification into the claims."). "[E]ven where a patent describes only a single embodiment, claims will not be read restrictively unless the patentee has demonstrated a clear intention to limit the claim scope . . . ." *Innova/Pure Water,*

Inc. v. *Safari Water Filtration Sys., Inc.*, 381 F.3d 1111, 1117 (Fed. Cir. 2004).

On the other hand, "[a] patent's specification provides necessary context for understanding the claims," and "sometimes the specification offers practically incontrovertible directions about claim meaning." *Abbott Labs,* 566 F.3d at 1288. When a patent specification

> makes clear that the invention does not include a particular feature, that feature is deemed to be outside the reach of the claims of the patent, even though the language of the claims, read without reference to the specification, might be considered broad enough to encompass the feature in question.

SciMed Life Sys., Inc. v. *Advanced Cardiovascular Sys., Inc.*, 242 F.3d 1337, 1342 (Fed Cir. 2001).

The prosecution history can also inform claim meaning "by demonstrating how the inventor understood the invention and whether the inventor limited the invention in the course of prosecution, making the claim scope narrower than it would otherwise be." *Phillips*, 415 F.3d at 1317.

Courts may also consider "extrinsic evidence," which is evidence outside of the patent and prosecution history, including dictionaries. *Id.* at 1318. Although "extrinsic evidence can help educate the court regarding the field of the invention and can help the court determine what a person of ordinary skill in the art would understand the claim terms to mean," *id.* at 1319, it is "less reliable than the patent and its prosecution

history," *id.* at 1318. Thus, extrinsic evidence is "unlikely to result in a reliable interpretation of patent claim scope unless considered in the context of the intrinsic evidence." *Id.* at 1319.

Furthermore, patent law requires that claims be "definite." This requirement stems from the language of 35 U.S.C. § 112(b), which states "[t]he specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention." "A claim is indefinite if its legal scope is not clear enough that a person of ordinary skill in the art could determine whether a particular [product] infringes or not. *Geneva Pharmaceuticals, Inc.* v. *GlaxoSmithKline PLC*, 349 F.3d 1373, 1384 (Fed Cir. 2003). A claim is not indefinite merely because it is difficult to construe. Rather, a claim is only indefinite if it is so "insolubly ambiguous" that is it simply "not amenable to construction." *Datamize LLC* v. *Plumtree Software, Inc.*, 417 F.3d 1342, 1346 (Fed. Cir. 2005). Even if reasonable minds might differ regarding the proper construction, the claim will be definite if it can "can be given any reasonable meaning." *Id.* at 1347; *see also Exxon Res. & Eng'g Co.* v. *United States*, 265 F.3d 1371, 1375 (Fed. Cir. 2001). According to the Federal Circuit, examples of when a claim may be indefinite include claims "completely dependent on a person's subjective opinion," claims

without a "proper antecedent basis where such basis is not otherwise present by implication or the meaning is not reasonably ascertainable," and claims that "include[]a numeric limitation without disclosing which methods of measuring . . . should be used." *Halliburton Energy Servs., Inc.* v. *M-I LLC*, 514 F.3d 1244, 1249 (Fed. Cir. 2008).

III. CLAIM CONSTRUCTION

A. The '924 Patent

1. Preamble - "Debugging"

Appthority argues that the preamble to Claim 1 is limiting. Veracode disagrees. I find that the preamble merely provides the context and purpose for the claim and therefore does not constitute a substantive limitation.

The traditional rule is that preamble language does not act as a limitation unless it is "necessary to give life, meaning, and vitality to the claim." *Kropa* v. *Robie*, 187 F.2d 150, 152 (C.C.P.A. 1951). When limitations in the body of the claim "rely upon and derive antecedent basis from the preamble, the preamble may act as a necessary component of the claimed invention," but "if the body of the claim sets out the complete invention, the mythed language of the preamble may be superfluous." *Eaton Corp.* v. *Rockwell Int'l Corp.*, 323 F.3d 1332 (Fed. Cir. 2003) (quoting *Schumer* v. *Lab. Computer Sys. Inc.*, 308 F.3d 1304, 1310 (Fed. Cir. 2002). In other words, if the body of the claim recites a

"structurally complete invention" without reference to the preamble, the preamble does not limit the claim. *Catalina Marketing Int'l, Inc.* v. *Coolsavings.com, Inc.*, 289 F.3d 801, 808 (Fed. Cir. 2002). The body of Claim 1 in the '924 Patent recites a structurally complete invention.

Without the preamble, Claim 1 reads:

> an analyzer for causing die [sic] computer to statically analyze a representation of the binary program file in order to detect the presence of program errors or potential program errors in the representation of the binary program file without executing the binary program file, wherein the representation of the binary program file is an intermediate file; and
>
> an output arrangement for causing the computer to output an error list of the errors or potential errors detected by the analyzer.

('924 Patent, 17:13-22.) This language presents the complete idea without any need to reference the preamble. It does not refer to any antecedent or idea defined only in the preamble, nor does it lack clarity in the absence of the preamble. The claim is straightforward: an analyzer that detects errors in a static, intermediate representation of binary code and outputs those errors into a list. All of this information appears is the language of the body of claim itself.

The preamble merely adds that Claim 1 describes

> a static debugging tool for use with a computer and for debugging a binary program file without requiring the execution of the binary program file in order to detect the presence of program errors and potential program errors.

(*Id.* at 17:9-13.)

Appthority argues that the term "debugging" is "necessary to give life, meaning, and vitality to the claim," *Kropa*, 187 F.2d at 152, and, therefore, that it substantively limits the claims and that the court must construe it. I disagree. Even if the body of Claim 1 did not clearly indicate to a person of ordinary skill in the art that this analyzer constitutes a "debugging tool," (*Id.* at 17:9) - a dubious proposition in light of the claim language stating that the analyzer "detects the presence of program errors or potential program errors," (*Id.* at 17:15-16.) - the issue is ancillary. The language of the body of the claim describes what the patent claims: an analyzer to find errors in static, intermediate representations of binary code. That the preamble describes such an analyzer as a "debugging tool" neither adds to nor takes away from what the patent claims. It is merely shorthand. It succinctly describes the purpose of the analyzer, but descriptions of purpose in a preamble are not limiting. *Catalina Mktg. Int'l*, 289 F.3d at 808 ("[A] preamble is not limiting where a patentee . . . uses the preamble only to state a purpose or intended use for the invention." (internal quotations omitted)).

Unlike in *Vizio, Inc.* v. *Int'l Trade Comm'n* and *Griffin* v. *Bertina*, where the Federal Circuit held that the phrases "for decoding" and "for diagnosis" were substantive limitations rather

than mere statements of purpose, revealing the errors or potential errors in a binary file has more directly practical use. In *Vizio* and *Griffin*, the Federal Circuit held the preambles to be limiting because, without those descriptions, the claimed processes would be "mere[] academic exercises," *Vizio*, 605 F.3d 1330, 1341 (Fed. Cir. 2010); *Griffin*, 285 F.3d 1029, 1033 (Fed. Cir. 2002). The list of errors and potential errors that the analyzer described in Claim 1 produces is analogous to the phrase "for diagnosis" in *Griffin*, not the underlying, purely academic "obtaining nucleic acid and assaying for a point mutation." *Griffin*, 285 F.3d at 1033. The list of errors essentially is a diagnosis for what is wrong with the analyzed software. It does not require any further description of purpose to give life, meaning and vitality to the claim. The described list of errors in the '924 Patent is useful, as in the *Griffin* patent, "for diagnosis;" in this case, for diagnosis of the analyzed program. Therefore the phrase "debugging tool" merely describes the purpose of the structurally complete analyzer and is not necessary to give life or purpose to the claim as the phrases "for decoding" and "for diagnosis" were in *Vizio* and *Griffin.*

Defendant's argument that the body of Claim 1 is not structurally complete because it does not describe any specifics about the way in which the analyzer works is little more than

distraction. Defendant's argument ignores the 23 other claims in the Patent - including four dependent claims specifically addressed to Claim 1 - that describe the analyzer in greater detail. If courts required each individual claim in a patent to set out every detail of the entire patent in order to define a structurally complete invention, no valid patent could ever have more than a single, unwieldy claim. I find that the preamble does not limit Claim 1, and therefore decline to construe the term "debugging."

2. "Program Errors"

Appthority argues that the term "Program Errors" means "unintended programming mistakes inconsistent with the program's intended design." It also offers the alternative, "a value or condition that is not consistent with the true, specified, or expected value or condition." Veracode argues that the term's meaning is clear and does not require construction. However Veracode also contends that the plain meaning of "Program Errors" includes intentional errors that may be consistent with the intended design of the analyzed program. While this is a potentially appropriate construction of the term "error," it is not likely to be intuitive to a jury, nor would declining to construe the term resolve the crux of the disputes between the parties: whether an error must be unintentional and whether it must be contrary to the program's intended design. The term

requires construction to resolve this dispute and to clarify the term's meaning for a jury. *See O2 Micro Int'l* v. *Beyond Innovation Tech.*, 521 F.3d 1351, 1362-63 (Fed. Cir. 2008) ("When the parties present a fundamental dispute regarding the scope of a claim term, it is the court's duty to resolve it.").

The word "error" does not necessarily imply "mistake." Although these two words have overlapping definitions, they are not entirely synonymous. An error can be intentional, a mistake cannot. The '924 Patent does not explicitly define the word "error." Therefore, recourse to a technical dictionary for the relevant art may be appropriate if that definition is consistent with the Patent's use of the term. *MIT* v. *Abacus Software*, 462 F.3d 1344, 1351 (Fed Cir. 2006) (holding that "it is appropriate for us to look to dictionary definitions of the terms" where "the specification does not define the term" and "the most that can be said is that the specification is not inconsistent" with the proposed constructions by the parties).

The Microsoft Press Computer Dictionary from the time the '924 Patent issued defines "error" as "[a] value or condition that is not consistent with the true, specified, or expected value or condition. In computers, an error results when an event does not occur as expected or when impossible or illegal maneuvers are attempted." (Def.'s Opening Br., Ex. A at 179.) An "error" can be *either* an unexpected occurrence *or* the result

of an impossible or illegal maneuver. Thus, an error is not necessarily a mistake, nor is it necessarily contrary to the purpose of a program.

There are a variety of conceivable reasons why a software developer might purposefully include a function that would return an impossibility or conduct an illegal maneuver in order to advance the purposes of the program. Indeed, the definition: "the result of an impossible or illegal maneuver" is consistent with the way in which the specification uses the term "error." The specification provides a variety of examples of the kinds of errors that the '924 Patent is designed to detect. These include "uninitialized memory, array bounds violations, accesses outside of allocated memory, inconsistent argument types or returns between calls and called functions, . . . invalid references to automatic memory . . ., accessing freed memory," ('924 Patent at 9:3-14), as well as "potential return of an automatic address," (*Id.* at 9:38-39), and "function parameters that are out of bounds, memory access errors, and potential invalid operands for arithmetic operations such as dividing by zero," (*Id.* at 16:35-37). Some of these examples are necessarily unintentional. Others may be intentional and designed to further some purpose of the program. Many of the examples of errors listed in the specification - such as accessing invalid memory and array bounds violations - are useful tools for exploitation or subversion.

For instance, Plaintiff's expert, Aviel Rubin, describes how some software developers might purposefully include an array bounds violation in order to maintain a "backdoor" into the program. Intent cannot logically be a limitation on the patent's use of the term "error" where the specification itself uses the term to mean errors that can be either intentional or unintentional.

It is not relevant for purposes of claim construction that Rubin's example of a potentially intentional error also potentially constitutes malicious code. The relevance is that the specification includes, as examples of potential errors that the patented analyzer detects, certain errors that might be either intentional or unintentional. The '924 Patent does not detect only unintentional errors contrary to the purposes of the program - some errors it detects may be intentional. Nor does it specifically detect malicious code - some of the errors it detects are, of necessity, mistakes that are neither malicious nor intentional. Instead, the common element in all of the errors the '924 Patent's specification lists is that they are all the result of an illegal or impossible maneuvers in the code.

Plaintiff's argument - that Appthority lists many examples of errors set out in the specification that must be unintentional whereas Veracode lists only one or two examples of errors in the specification that can be intentional - is perplexing. If even a single example of an error listed in the specification can be

intentional, then, as a matter of logic, the term is not necessarily limited to unintentional errors. Because the Patent does not limit this term to unintentional errors, I also reject Veracode's amended proposed construction suggested in supplementary briefing: "a value or condition that is not consistent with the true, specified, or expected value or condition." This portion of the MS Computer Dictionary definition focuses on the unexpected nature of certain values or conditions, which is not a proper limitation on the scope of the term "Program Error" as used in the '924 Patent. The common factor in each of the listed examples of errors in the specification is that they are the result of "impossible or illegal maneuvers." I am mindful, however, that the infringement contentions in this case involve the ability of each party's software to detect nefarious, illegal activity - activity that is illegal because the law prohibits it. The phrase "the result of an illegal or impossible maneuver," by contrast, uses the word "illegal" differently - to mean invalid.

I construe "Program Errors" to mean "the result of an invalid or impossible maneuver."

3. "Intermediate File"

The parties agree that the term "Intermediate File" means a "representation of the binary file." However, the parties dispute (a) whether it must be machine independent, meaning that

the code is not unique to the processor's specific binary language, and (b) whether it is, by definition, not source code. Appthority proposes the construction "machine independent representation of the binary file." In its supplementary briefing, Appthority also suggests the alternative, "a representation of the binary file for analysis by the analyzer." Veracode proposes the construction "a representation of the binary file that is not source code."

a. Machine Independent

The "Independent File" is not necessarily machine independent. The Patent claims repeatedly describe the intermediate file as a "machine independent, intermediate file" However, machine independence is not an inherent characteristic of the Patent's use of the term "Intermediate File."

Claim 1 describes a debugging tool that analyzes a representation of the binary program file "wherein the representation of the binary program file is an intermediate file." ('924 Patent, 17:18-19.) Claim 2, which depends on Claim 1, states that a decompiler causes the computer to "translate the binary program file into an intermediate, machine independent program file." (*Id.* at 17:26-28.)

Similarly, Claim 13 describes a method of detecting program errors "wherein the representation of the binary program includes an intermediate file that represents the binary program file and

includes the flow paths and flow structure associated with the binary program." (*Id.* at 18:19-23.) Claim 15, which depends on Claim 13, claims "a method according to Claim 13 further comprising the step of decompiling the binary program file to create a machine independent, intermediate file to be analyzed in the analyzing step." (*Id.* at 18: 37-40.) Claims 17 and 18 also present the same relationship.

Under principles of claim differentiation, Claims 1 and 2 must be distinct from one another, as must Claims 13 and 15. *See Comark Comm'ns* v. *Harris Corp.*, 156 F.3d 1182, 1187 (Fed. Cir. 1998)(the doctrine of claim differentiation "create[s] a presumption that each claim in a patent has a different scope."). Furthermore, The independent claims - Claims 1 and 13 - must be broader than the dependent claims - Claims 2 and 15. *Am. Medical Sys., Inc.* v. *Biolitec, Inc.,* 618 F.3d 1354, 1360 (Fed. Cir. 2010) ("Under the doctrine of claim differentiation, those dependent claims give rise to a presumption that the broader independent claims are not confined to that range.").

Appthority argues that Claims 1 and 2 are distinguishable on bases other than the machine independence of the intermediate file described in Claim 2. For instance, Claim 2 recites a "decompiler" not present in Claim 1. Claims 13 and 15 are susceptible to the same distinction. Claim 13 states that the claimed "method of detecting program errors" includes analyzing

an intermediate file, which is a representation of the binary file. Claim 15 recites a further step of decompiling the binary file into a machine independent intermediate file. The same is also true of Claim 18, which uses the phrase "machine independent program file" and which depends on Claim 17. Although the doctrine of claim differentiation may be satisfied by the single, narrow distinction of decompilation, *see Mantech Envr. Corp.* v. *Hudson Envr. Servs.,* 152 F.3d 1368, 1376 (Fed. Cir. 1998), the Patent's consistent use of the phrase "machine independent" only in dependent claims and specific embodiments makes clear that the Patent intends that the term "Intermediate File" should not be limited to machine independence. Thus, although the doctrine of claim differentiation does not apply in its strictest sense because there are two grounds on which to distinguish Claim 2 from Claim 1, Claim 15 from Claim 13, and Claim 18 from Claim 17, it remains instructive in illuminating the intended scope of the term "Intermediate File."

The phrase "machine independent" appears nine times throughout the '924 Patent. However, it neither appears in a high-level description of what the Patent claims, nor does it appear in any independent claim. It first appears in the "Summary of Invention section" in the description of a preferred embodiment. ('924 Patent at 1:68.) However, this immediately follows a broader description of what the patent claims,

including an explanation that the invention analyzes a "representation of the binary file." (*Id.* at 1:50-63.) It next appears several times throughout the section of the Patent entitled "Detailed Description of the Preferred Embodiments," (*See id.* at 3:61, 63; 4:63; 6:1, 10), however, the Federal Circuit has repeatedly asserted that reading limitations from particular embodiments into the claims is the "cardinal sin" of claim construction. *See Phillips* v. *AWH Corp.*, 415 F.3d 1303, 1319-20 (Fed. Cir. 2005) (citing *SciMed Life Sys.* v. *Adv. Cardiovascular Sys., Inc.,* 242 F.3d 1337, 1340 (Fed. Cir. 2001)). Finally, it appears three times in the claims themselves, in Claim 2, Claim 15, and Claim 18. However, each of these claims depends on an earlier independent claim, which also uses the phrase "Intermediate File," but does not include the limitation "machine independent."

The independent claims all use the term "Intermediate File," while their respective dependent claims use the more specific phrase "machine independent intermediate file." Because the doctrine of claim differentiation suggests that an independent claim should be broader than its dependent, and because of the direct connection between the use of the broad phrase "Intermediate file" in the independent claims and the use of the more specific phrase "machine independent intermediate file" in the dependent claims, I find that the independent claims

contemplate intermediate files that are not necessarily machine independent despite the fact that there may also be other grounds to differentiate the claims. *See Phillips*, 415 F.3d at 1315 ("[T]he presence of a dependent claim that adds a particular limitation gives rise to a presumption that the limitation in question is not present in the independent claim.").

b. Not Source Code

By the very nature of the term "Intermediate File," it cannot be source code. Nor can it be binary code. It is "intermediate" because it is *between* the binary file and the source code. This meaning of the term "intermediate" is well understood by those of ordinary skill in the art of software development. The Microsoft Press Computer Dictionary from the time period when the '924 Patent issued defines "intermediate language" as "[a] computer language used as an intermediate step between the original source language, usually a high-level language, and the target language, usually machine code." By extension, an "Intermediate File" is a file, which is written in the intermediate language, containing a representation of the binary program file, which was written in binary code.

This is equally clear from the '924 Patent itself. In describing the background of the invention, the Patent distinguishes the prior art, stating, "[p]resently, virtually all debugging tools do their debugging at the source code level."

('924 Patent at 16-18.) Then, in describing how the invention improves on this prior art, the Patent summary states that, "[i]n one embodiment of the present invention . . . an intermediate machine independent program file . . . serves as the representation of the binary program file to be analyzed . . . ." (*Id.* at 65-69.) As one of the stated improvements over the prior art, the described invention analyzes the intermediate file rather than source code. It cannot be the source code because the stated improvement from the prior art is that the '924 Patent can analyze a program without the effort of translating it fully back into source code. The intermediate file also cannot *be* the binary program file since it is a *representation* of that file. Rather the "Intermediate File" is a representation of the program's code written in a higher-level language than binary, machine code, but in a lower-level language than full source code.

Appthority's position - that the intermediate file *can* be source code and that it is only intermediate in the sense that is a stepping stone in the debugging process - is a distinction without a difference. Its description of the intermediate file as a stepping stone between the binary program file and the ultimate list of errors is entirely consistent with Veracode's proposed construction. Appthority has not articulated any way in which the intermediate file could be considered a stepping stone

in the debugging analysis other than as an intermediate language between binary and source code. Thus, its alternative proposed construction, "a representation of the binary file for analysis by the analyzer," obscures rather than clarifies. It does nothing to resolve the parties dispute regarding source code and would therefore be an inappropriate construction.

Appthority also argues that the "Patent makes no statement at all about what *sort* of code the intermediate file contains (binary vs. source)," but this argument simply assumes what it seeks to prove: that the intermediate file is intermediate in some way other than the kind of code it contains. I therefore reject Defendants argument, and ultimately find that the intermediate file is neither source code nor binary code.

I construe "Intermediate File" to mean "a representation of the binary file that is neither in binary code nor source code."

4. "Decompiler"

The parties' disputes regarding construction of the term "Decompiler" echo their previous disputes over the meaning of "Intermediate File." Appthority proposes the construction, "[Tool for] converting binary code to machine independent, high-level language source code" Veracode proposes the construction, "Software that translates the binary program file into an intermediate, machine independent program file." I find that the term "Decompiler," as used in the '924 Patent does not generate

high-level source code; the code it generates is machine independent.

a. High-Level Language Source Code

Claim 2 recites a "decompiler for causing the computer to decompile the binary program file and translate the binary program file into an intermediate, machine independent program file." ('924 Patent at 17:25-28.) As discussed above, *see* Section III(A)(3)(b), the intermediate file that this decompiler generates is neither source code nor binary code. I reject Appthority's attempt to graft the term "source code" onto "Decompiler" for the same reasons I rejected its attempt to do the same to the term "Intermediate File."

Appthority's citation to the MS Computer Dictionary to support its proposed construction is unavailing. Although the MS Computer Dictionary defines "Decompiler" as "a program that attempts to generate high-level source code from assembly language code or machine code," that dictionary definition contradicts the way in which the Patent uses the term, and the intrinsic evidence of the Patent controls over the extrinsic evidence of a dictionary definition. *See W.E. Hall Co.* v. *Atlanta Corrugating, LLC*, 370 F.3d 1343, 1350 (Fed. Cir. 2004) ("While dictionaries may be used to ascertain the plain and ordinary meaning of claim terms, the intrinsic record is used to resolve ambiguity in claim language or, where it is clear, trump

inconsistent dictionary definitions.”). A Patentee is free to use or define a term in a way that does not necessarily comport with the conventional meaning of that term. *See Honeywell Int’l, Inc.* v. *Universal Avionics Sys. Corp.*, 493 F.3d 1358, 1361 (Fed. Cir. 2007)(“When a patentee defines a claim term, the patentee’s definition governs, even if it is contrary to the conventional meaning of the term.”). Indeed, the very nature of Patents, often requires defining new terms or reconsidering the meaning of existing terms in order to capture a new method or device not yet understood by the prior art. The claims in the ‘924 Patent that use the term “Decompiler” state that the purpose of decompilation is to create an intermediate file, and any appropriate construction of that term must comport with the way the patent uses it. (*See* ‘924 Patent, Claims 2, 15, 16, 18.)

In its supplementary brief, Appthority argues that this Court’s construction of Decompiler in combination with its construction of Intermediate File would have the unintended effect of excluding source code from the meaning of the Decompiler, contradicting the ordinary industry meaning of the term. To the contrary, this exclusion is conscious, intentional, and necessary in light of the language of the specification. As discussed above, the ‘924 Patent does not use the word “Decompiler” according to the industry-standard definition of a tool to translate low-level machine code into high-level source

code. Rather, the Patent uses the term Decompiler more generically as a tool that translates a lower-level language into a higher-level form: in this case, from machine code into an intermediate file. The Patent's specific use of a term - and not the industry standard understanding - controls the appropriate construction of any term. *See Honeywell Int'l, Inc.*, 493 F.3d at 1361. The intrinsic evidence - the language of the patent - uses the term Decompiler in a way that is inconsistent with translation into source code and therefore Appthority's reliance on contrary extrinsic evidence - industry dictionaries and understanding - is misplaced. *See id.*

Appthority also argues that Veracode's proposed construction renders a step of the claimed decompilation process superfluous. This cannot be reconciled with either the plain claim language or Appthority's own proposed construction. Claim 2 recites "a decompiler for causing the computer to decompile the binary program file *and* translate [it] into an intermediate, machine independent program file." (*Id.* at 17:25-28 (emphasis added).) Appthority argues that the word "and" indicates that "decompiling" and "translating" are two separate and distinct steps. However, reading the claims in their entirety, it is clear that the Patent uses the word "and" to signify that the language that follows defines "decompile." For instance, Claim 15 states a "method . . . of decompiling the binary program file

to create a machine independent, intermediate file to be analyzed in the analyzing step," only reciting a single step. (*Id.* at 18:37-40 (emphasis added).) The phrases "a decompiler *for* causing" and "decompiling . . . *to* create" equate the decompiler or decompilation process with the creation of the intermediate file, essentially defining "Decompiler" as the tool for executing that process. Thus, reading Claim 15 and Claim 2 together, the meaning of the word "and" in Claim 2 becomes clear. It equates decompiling the binary program file with translating it into the intermediate file. It does not signify two separate steps, as Appthority contends. In fact, Appthority's proposed construction of "Decompiler" would render the second half of the language in Claim 15 redundant. Furthermore, Appthority undercuts its own argument that Claim 2 must recite two steps because Appthority itself proposes one-step construction of "Decompiler." Thus Appthority's argument runs contrary to the claim language itself, and runs afoul of the "'bedrock principle' of patent law that 'the claims of a patent define the invention to which the patentee is entitled the right to exclude.'" *Phillips*, 415 F.3d at 1312 (quoting *Innova/Pure Water, Inc.* v. *Safari Water Filtration Sys., Inc.*, 381 F.3d 1111, 1115 (Fed. Cir. 2004)).

Because the Patent and the MS Dictionary itself make clear that the intermediate file is not source code, *see* Section III(A)(3)(b), the term "Decompiler," as used in the '924 Patent,

is inconsistent with translation into high-level source code. I therefore decline Appthority's invitation to graft the phrase "source code" onto the meaning of "Decompiler."

b. Machine Independent

Both proposed constructions include the limitation that the Decompiler creates a *machine independent* program file. I agree with this aspect of the construction. Unlike the term "Intermediate File," which appears in the Patent's independent claims, the term "Decompiler" or "decompile" appears in the dependent claims alongside the explicit "machine independent" limitation. (*See* '924 Patent, Claims 2, 15, 18.)[1] "Decompiling" appears in Claim 16, but Claim 16 depends on Claim 15, which itself describes the "Intermediate File" as machine independent, and as a dependent claim, Claim 16 incorporates the limitations of the independent claim on which it depends. Thus, the Patent claims themselves indicate that machine independence is necessarily a limitation on the term "Decompiler," as the parties themselves concede.

I construe "Decompiler" as "a tool for translating the binary program file into an intermediate, machine independent program file."

---

[1] "Decompiled" also appears in Claim 3, but Claim 3 does not discuss the "Intermediate File," and so does not counsel in favor or against limiting the decompiler to producing machine independent products.

5. “Determining and Symbolically Representing the Function Flow”

The parties agree that the words “function flow” describe “how functions are associated and interconnected with other functions.” In their briefs, the parties disputed whether the term “Symbolically Representing” is necessarily limited to “graphical” diagrams such as Binary Decision Diagrams (“BDD”), or whether it can include other forms of symbolic representation. At the claim construction hearing, I proposed the construction “identifying how functions are associated and interconnected with other functions and representing those associations and connections through symbols.” The parties have represented that they both accept to this construction.

B. The ‘609 Patent

1. “Optimized”

The parties agree that the ‘609 Patent does not use the term “Optimized” as it is otherwise generally understood within the art. Within the art of software development, “Optimized” traditionally means that the program is streamlined and made more efficient. The ‘609 Patent’s use of the term “Optimized” is not consistent with this definition. For instance, the phrase “processing the executable code to generate an optimized, exhaustive control flow model,” (‘609 Patent at 15:4-5), implies that the optimization process ultimately results in a larger, more complete file rather than a smaller or faster file.

Appthority argues that the term "Optimized" means "iteratively refined by repeat analysis of data flow and control flow until the model is complete and as effective as possible." Veracode argues that "Optimized" means "refined."

Veracode's proposed construction is insufficient. Replacing "Optimized" with "refined" adds no clarity for the jury and resolves no dispute. It would substitute one ambiguous term for another. In fact, it would impermissibly substitute a more general term for a more specific one. "Refined" simply means "improved" or "made better" without necessarily implying any particular form of improvement. "Optimized" implies that something is transformed into its "best" form. Thus, "Optimized" - meaning "made best" - is actually a subset of "refined" - meaning "made better." The word "refined" would conform to the Patent's use of the term "Optimized," (*see, e.g.*, '609 Patent at 6:61-63 ("[A]n optimized (refined) model is produced")), but it would add definitional value to the construction of the term "optimized." *See Network, LLC* v. *Centraal Corp.,* 242 F.3d 1347, 1352 (Fed. Cir. 2001) ("The role [of claim construction] is neither to limit nor to broaden the claims but to define, as a matter of law, the invention that has been patented."). Rather, through reference to other claim language, the specification, and the prosecution history, a person of ordinary skill in the art would be able to glean the intended meaning of "optimized."

The disputes center around two aspects of Appthority's proposed construction: (a) whether the process of optimization in the '609 Patent is necessarily iterative, and (b) how to characterize the resulting optimized models.

a. Iteration

The specification and prosecution history of the '609 Patent make clear that iteration is an inherent aspect of the optimization process. The written description describes how the invention creates "an optimized (refined) model." ('609 Patent at 6:63.) It states that

> a first, fitting . . . model is approximated . . . and then iteratively improved to form a refined . . . model representing all necessary states and branches within the code. This double loop occurs, in sequence, throughout the program, iterating . . . until an optimized (refined) model is produced.

(*Id.* at 6:56-63.) Although this language appears in the specification and cannot, by itself, operate to limit the claim language, the patent prosecution history confirms that this iteration process is inherent in the claims and is not simply limited to a preferred embodiment.

In the prosecution history of the parent to the '609 Patent, the inventor distinguished prior art (the '924 Patent) stating that independent Claim 1 of the '609 Patent, describes a method whereby "[a] fitting data flow model is identified . . . which is then optimized . . . to form a refined data flow model; propagated as a function of the respective defined data flow

until substantially all data variables are modeled." U.S. Patent App. Serial No. 10/314,005, Amendment (Sept 29, 2005) at 12-13. The prosecution history uses practically identical language to describe the propagation of the control flow model. *See id.* Veracode argues that this language describes the optimization process without reference to iteration, but that appears to misunderstand the language. Although the quoted language does not use the words "iteration" or "iterate," and is not entirely clear, it seems to describe an iterative process. The data flow model is "propagated as a function of the . . . data flow *until* . . . all . . . variables are modeled." *Id.* (emphasis added). The word "until" implies that the propagation process does not merely occur once, but either repeats or continues. The fact that this process creates a data flow model using a function of the data flow itself more than once suggests that the process is iterative or recursive, recursion being a specific form (and therefore a subset) of iteration. Nothing in the independent claim language using the term "Optimized" implies that the process must specifically be recursive, but Dependent Claims 2 and 4 specifically use the term "recursion." Thus, under the doctrine of claim differentiation, the independent claims must have the broader reach of iteration as compared with the dependent claims, which specify a particular form of iteration: recursion.

b. Characterization of the Optimized Models

Appthority's proposed phrase, "until the model is complete and as effective as possible" is not tethered to any use of the term "Optimized" anywhere in the Patent. The word "effective" never appears in the patent at all. The Claims use the word "complete" only to describe the intermediate representation, (*see* '609 Patent at 15:6-9, 16:31-34), but not to directly describe the optimized model itself.

Rather, the '609 Patent characterizes the optimized models in the specification where it says that the "model represent[s] all necessary states and branches within the code" and then states that the "double loop occurs . . . until [the] optimized (refined) model is produced." The prosecution history also characterizes the optimized models, stating that Claim 1 recites "a fitting data flow model . . . which is then optimized . . . until substantially all data variables are modeled." U.S. Patent App. Serial No. 10/314,005, Amendment (Sept 29, 2005) at 12-13. It characterizes the optimized control branch model using the same language as in the data flow model. *Id.* Because the language in the specification may describe only one particular embodiment, while the prosecution history characterizes the claim language itself, I find that the appropriate characterization of the optimized model is that it models "substantially all data variables or control branches."

I construe “Optimized” to mean “refined by iteration until substantially all data variables or control branches are modeled.”

2. “Exhaustive”

Appthority argues that “Exhaustive” means “testing all program possibilities, or considering all program elements, from entry to exit.” Veracode argues that the meaning of the term “Exhaustive” is clear and that it requires no construction. I agree with Veracode that the term “Exhaustive” does not require construction.

The term “Exhaustive” appears in the claims only and does not appear in the specification. Unlike its approach to the term “Optimized,” the Patent uses “Exhaustive” according to its plain, ordinary meaning. The Patent neither uses the word contrary to its plain meaning, nor is it such a technical or ambiguous word that the jury will have trouble understanding it. There is, therefore, no reason to resort to the complicated definition Appthority proposes. *See Hoover Grp., Inc.* v. *Custom Metalcraft, Inc.*, 66 F.3d 299, 304 (Fed. Cir. 1995) (applying the ordinary meaning of a disputed term where “nothing in the specification, prosecution history, or prior art to suggest other than the ordinary meaning”).

Appthority argues that the focus and overall thrust of the Patent language, which emphasizes completeness, counsels in favor

of construing the term, and cites to the Miriam-Webster Online Dictionary to support its proposed language, which does not appear anywhere in the language of the Patent itself or the Patent's prosecution history. This is not a permissible form of construction. *See Allen Eng. Corp.* v. *Bartell Indus., Inc.*, 299 F.3d 1336, 1345 (Fed. Cir. 2002) ("It is well settled that "there is no legally recognizable or protected essential element, gist or heart of the invention in a combination patent. Rather, the invention is defined by the claims." (internal citations, quotations, and alterations omitted)). A party may not derive its construction from the overall gist or thrust of the patent, but must ground its construction in the language of the patent itself.

I decline to construe the term "Exhaustive."

3. "Data Flow Signatures"

The parties agree that "Data Flow" means the "process whereby variables or data storage elements are read from and/or written to memory." The parties also agree that in the context of the software development art and the '609 Patent, the term "Signatures" is synonymous with "patterns." Thus, Veracode argues that "Data Flow Signatures" means a "pattern of process whereby variables or data storage elements are read from and/or written to memory." However, Appthority argues that the term "Signatures" has no coherent meaning when appended to the term

"Data Flow" and therefore that the full term, "Data Flow Signatures," is insolubly ambiguous and indefinite. *See Datamize LLC* v. *Plumtree Software, Inc*., 417 F.3d 1342, 1346 (Fed. Cir. 2005). The parties have each submitted expert declarations in support of their respective positions.

The first problem with Appthority's argument that the phrase "Data Flow Signatures" is inherently nonsensical is that the phrase appears in various articles in the literature. For instance, an article entitled "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," published by professors at Duke, describes a method of checking for errors in a program by referring to "dataflow signatures." A. Meixner, Argus: Low-Cost, Comprehensive Error Detection in Simple Cases, Appearing in 40th Annual International Symposium on Microarchitecture (Dec. 2007) at 3-4. Similarly, an article in IEEE Security and Privacy entitled "Toward Application-Aware Security and Reliability" discusses a "dataflow signature checking" technique which "encod[es] the instructions that write to the critical object as a signature and then check[s] the signature at runtime." R. K. Iyer et al., Toward Application-Aware Security and Reliability, IEEE Security & Privacy (Jan./Feb. 2007) 57, at 59-60. In other words, when the program runs, it checks a previously recorded "signature" of the dataflow. This use is consistent with the '609 Patent's use of the term "Data Flow Signatures." Although

such extrinsic evidence is not dispositive of a term's definiteness, it demonstrates that Appthority's primary contention - that the term simply does not make sense in the art - does not withstand scrutiny. The intrinsic evidence also indicates that the term is definite.

In the context of the '609 Patent, "Signatures" refer to patterns of code that the invention can use to identify a stock function within the code being analyzed. The Patent incorporates an article by reference which explains that "patterns were previously known as signatures in the . . . literature." (*See* '609 Patent at 12:2-6 (citing M. Van Emmerik, "Signatures for Library Functions in Executable Files Using Patterns," Proceedings of the 1998 Australian Software Engineering Conference, Adelaide, 9-13 November, 1998, IEEE-CS Press, pp. 90 n. 1).) Claim 1 of the '609 Patent states that the invention "process[es] the executable software code to generate an optimized, exhaustive data flow model . . . ." ('609 Patent at 14:66-67.)

Claim 11, which depends on Claim 1 describes the invention's method of generating the exhaustive data flow model, stating that it "generat[es] data flow signatures for the executable software code; and compar[es] the generated data flow signatures to one or more predefined data flow signatures, each representing a data flow model." (*Id.* at 15:51-16:3.) In simpler terms, part of the

way in which the ‘609 Patent generates the exhaustive data flow model is by seeking out patterns in the way pieces of the analyzed code read and write data to memory that correspond to known patterns in order to identify known models for those pieces of code.

Appthority’s expert, Dr. Paul Clark, suggests that “Data Flow Signatures” is indefinite because software fingerprinting techniques, such as cyclic redundancy codes (“CRC”s), must apply to particular pieces of data, not “data flow.” He also contends that it is not possible to compare patterns of such specific data against known libraries because they will be located in ranges of code specific to that piece of software and are therefore not amenable to comparison with standard models.

However, Dr. Clark’s Declaration misunderstands the purposes of data flow signatures in the ‘609 Patent. The Patent does not mention CRCs or the other signature techniques Dr. Clark references. Furthermore, as Dr. Aviel Rubin, Veracode’s expert, explains, the CRCs and the signature techniques Dr. Clark discusses are irrelevant to the ‘609 Patent because they test data integrity - ensuring that the particular data in one place is the same as data in another location after the program transmits that data. They do not, as the Patent calls for, identify a pattern of reading and writing data that might be associated with a known model. Because the CRC and other data-

integrity techniques Dr. Clark discusses are not relevant to the '609 Patent, the fact that they cannot apply to "data flow" does not render the claim indefinite.

Dr. Clark also contends that the concept of "Data Flow Signatures" is nonsensical because the data produced by a program being analyzed are not drawn from general purpose external libraries, but are produced by the running of the application itself. He states that "an example of such data would be the input from the other code in the system being debugged" and that he "understand[s] this to be referred to as 'data flow' within the context of the patent." Again, this misunderstands the meaning of "Data Flow" in the context of the '609 Patent.

First and foremost, this argument is inconsistent with Dr. Clark's other argument. In paragraph 11 of his Declaration, discussed above, Dr. Clark assumes, correctly, that "Data Flow" refers to the process and flow of reading and writing data. However, in paragraph 10, he assumes, incorrectly, that "data flow" refers to the discrete data actually written by the program. Dr. Clark's analysis and "understanding" in paragraph 10 incorrectly and inapplicably refers to the data itself rather than the data flow despite his apparently correct understanding of data flow in paragraph 11.

Dr. Clark's argument in paragraph 10 is unpersuasive for the additional reason that the "pattern" that the '609 Patent seeks

to uncover is not in the substantive content actually written and read, but rather in the pattern of reading and writing - like the pattern of points where water flows into or out of a river, not the specific water flowing through it (data), or the path of the bends in the river (control flow). He may be correct that it would be nonsensical to attempt to identify a known model of a particular aspect of a program by looking at the data written by the analyzed program as it runs. This data is unique to the application and might not help to identify a particular model. However, the term "data flow" does not refer to the data itself, but to the pattern of writing to and reading from memory. It would be difficult to identify a river by looking solely at the molecules of water that flow through it, but much easier to identify it by looking at pattern of twists and turns (analogous to the control flow the Patent describes), or, more specific to this term: the pattern of the tributaries flowing into the river and the pattern of distributaries flowing out (analogous to the data flow the Patent describes.

The parties agree that the meaning of the term "Data Flow" is the "process whereby variables or data storage elements are read from and/or written to memory." I find that, in the context of the '609 Patent, the term "Data Flow Signatures" refers to the concept of patterns in this process, which, when compared to pattern libraries of similar data flows might help identify known

models for those programs. The term is therefore amenable to construction and neither insolubly ambiguous nor indeterminate.

I construe "Data Flow Signatures" to mean the "pattern of process whereby variables or data storage elements are read from and/or written to memory."

4. "Flaws"

Appthority argues that the term "Flaws" in the '609 Patent means "software errors i.e., 'unintended programming mistakes inconsistent with the program's intended design'" or should otherwise have the same construction as "Program Flaws" from the '924 Patent. Veracode argues that a jury can readily understand the term "Flaws" and that it therefore requires no construction.

Appthority's proposed construction is nearly identical to the one that it suggested - and that I have already rejected - for the meaning of the term "Program Errors" in the '924 Patent. (*See supra* Section III(A)(2).) It is even less appropriate here than it was as a proposed construction of "Program Errors." The term "Flaw" is broader than either "error" or "mistake." It allows for subjective deficiencies in quality, both intentional and unintentional, in addition to the objective inaccuracies captured by the words "error" and "mistake." Thus, in the context of the patents in suit, a "mistake" is a subset of "error," which, in turn, is a subset of "flaw." As I have already rejected Appthority's attempt to limit the term "error"

to mean “mistake,” I also reject its attempt to limit the broader term “Flaw” in the same way.

As with the term “Program Errors,” the crux of the dispute between the parties is whether a “Flaw” must be unintentional and whether it must be contrary to the program’s intended design. As in its proposed construction of “Program Errors,” discussed above, Appthority here relies on the MS Computer Dictionary definition of “error” despite a number of apparent deficiencies in such an argument, including that (1) Appthority does not cite to a dictionary definition of the term here under construction, “Flaw,” but to the definition of an entirely different word: “error;” (2) the word “error” does not appear anywhere in the ‘609 Patent; and (3) even if the word “error” were somehow relevant to construction of the ‘609 Patent, the cited definition says nothing about the two limitations Appthority seeks to impose: whether an error must be unintentional or whether it must be contrary to the purpose of the program.

Nothing about the plain meaning of the term “Flaw” implies that a flaw cannot be intentional or that a flaw must be contrary to the purposes of a program. In fact, the language in the abstract, the summary, and the specification indicate that the ‘609 Patent specifically contemplates the possibility of intentional flaws. The Patent Abstract states that “[t]he nano-code decompiler may be used to determine if flaws, security

vulnerabilities, or general quality issues exist in the code." ('609 Patent, Abstract.) The Patent summary states that the "decompiler may produce . . . a report showing the flaws, vulnerabilities, and/or poor programming practices in the original executable code . . . [including] flaws and pointers to badly constructed data structures, unchecked buffers, malicious embedded code or 'trap doors,' and the like." (*Id.* at 3:1-10.) Finally, the specification states that the "executable code can thus be scanned or analyzed for flaws or conditions, especially including security holes, bugger structure flaws exploitable via 'buffer overflow' attack, and other known and unknown risk factors." (*Id.* at 11:1-5.) Thus, the '609 Patent specifically contemplates security vulnerabilities including the buffer overflow attack that Dr. Rubin discusses by name and which he describes in his first Declaration as potentially intentional. The '609 Patent specifically contemplates security vulnerabilities, malicious embedded code, and other risk factors in the flaws it searches for, any of which may be part of the purpose of a given program and any of which may be intentional. I will not limit the term "Flaws" to exclude elements that the language of the Patent specifically includes.

Appthority's argues that I should limit the meaning of "Flaw" because the both the patent abstract and the summary list "flaws" as one of a variety of examples of programming problems,

including security vulnerabilities and general quality issues. Appthority reasons that under doctrines of Patent construction, "Flaw" must have a different meaning from the other examples in the list, such as "security vulnerabilities." *See Comark*, 156 F.3d at 1187 (when different words are used in separate claims, they are presumed to have different meanings). This argument fails for a number of reasons. First, the fact that the terms should not be construed as *synonymous* does not necessarily require that the terms be *mutually exclusive*. Second, Appthority offers no justification why any difference in meaning between "Flaw" and "security vulnerabilities" would require that flaws be unintentional or contrary to the purpose of a program. Finally, Appthority's argument ignores the plain language of the patent specification which uses the phrase "security holes" as an example of a "flaw." (*See* '609 Patent at 11:2-3.) The specification states that patented invention scans executable code for "flaws or conditions *including* security holes, buffer structure flaws exploitable via 'buffer overflow' attack, and other known and unknown risk factors." (*Id.* at 11:1-5 (emphasis added).) The specification specifically *includes* potentially intentional security holes and buffer structure flaws within the meaning of the term "Flaws."

Appthority suggests that security holes may be one of the "Conditions" rather than one of the "Flaws" mentioned, but this

also does not comport with the plain language. The phrase states that "flaws or conditions include[] security holes . . . and other known and unknown risk factors." This sentence utilizes parallel structure: A or B including X and Y. Parallel structure suggests that the first half of the phrase before "including" connects with the first half of the phrase after "including" and the same is true for the second half of each phrase. Thus, parallel structure suggests that the security holes and buffer structure flaws are "Flaws" and the "Conditions" are "other known and unknown risk factors." In fact, the patent specifically refers to buffer overflow attack - undeniably a security hole - as a buffer structure *flaw.* The mental gymnastics that Appthority suggests in order to distinguish between "Flaws" and security vulnerabilities does not comport with the plain claim language. The limitations Appthority seeks to impose on the term "Flaw" are not appropriate.

Unlike the term "Program Errors," as discussed above, *see* Section III(A)(2), the term "Flaw" does not require construction to resolve the parties' dispute. I have rejected the limitations Appthority seeks to impose on the term "Flaw," and the term does not have the same unintuitive or ambiguous meaning in the context of the '609 Patent that the term "Program Error" had in the context of the '924 Patent.

The meaning of the term "Flaw" is clear from the plain meaning of the word and its use in the Patent. I therefore decline to construe it.

IV. CONCLUSION

For the reasons set out at length above, I construe the terms of the various claims as set forth in this Memorandum, and as summarized in the Appendix, *see infra* Section V.

/s/ Douglas P. Woodlock
DOUGLAS P. WOODLOCK
UNITED STATES DISTRICT JUDGE

V. APPENDIX: CLAIM CONSTRUCTION SUMMARY

A. Patent No. 5,854,924

1. Phrase(s) in Patent	"Debugging"
Plaintiffs' Construction	[Preamble is not limiting]
Defendants' Construction	"Allowing Software developers to detect, locate, or correct logical or syntactical errors in or malfunctions of a program."
Court's Construction	[Preamble is not limiting]

2. Phrase(s) in Patent	"Program Errors"
Plaintiffs' Construction	[Does not require construction]
Defendants' Construction	"Unintended programming mistakes inconsistent with the program's intended design"
Court's Construction	"the result of an invalid or impossible maneuver"

3. Phrase(s) in Patent	"Intermediate File"
Plaintiffs' Construction	"A representation of the binary file that is not source code"
Defendants' Construction	"Machine independent representation of the binary file"
Court's Construction	"a representation of the binary file that is neither in binary code nor source code"

4. Phrase(s) in Patent	"Decompiler"
Plaintiffs' Construction	"Software that translates the binary program file into an intermediate, machine independent program file"
Defendants' Construction	"Tool for converting the binary code to machine independent, high-level source code"
Court's Construction	"A tool for translating the binary program file into an intermediate, machine independent program file"

5. Phrase(s) in Patent	"Determining and Symbolically Representing the Function Flow"
Plaintiffs' Construction	"Identifying and describing how functions are associated and interconnected with other functions"
Defendants' Construction	"Depicting how functions are associated and interconnected with other functions via binary decision diagrams (BDDs) or other graphical diagrams"
Court's Construction	"identifying how functions are associated and interconnected with other functions and representing those associations and connections through symbols"

B. Patent No. 7,752,609

1. Phrase(s) in Patent	"Optimized"
Plaintiffs' Construction	"Refined"
Defendants' Construction	"Iteratively refined by repeat analysis of data flow and control flow until the model is complete and as effective as possible"
Court's Construction	"refined by iteration until substantially all data variables or control branches are modeled"

2. Phrase(s) in Patent	"Exhaustive"
Plaintiffs' Construction	[Does not require construction]
Defendants' Construction	"Testing all program possibilities, or considering all program elements, from entry to exit"
Court's Construction	[Does not require construction]

3. Phrase(s) in Patent	"Data Flow Signatures"
Plaintiffs' Construction	"Pattern of processes whereby variables or data storage elements are read from and/or written to memory"
Defendants' Construction	[Indefinite]
Court's Construction	"Pattern of processes whereby variables or data storage elements are read from and/or written to memory"

4. Phrase(s) in Patent	“Flaws”
Plaintiffs’ Construction	[Does not require construction]
Defendants’ Construction	“Software errors, i.e., unintended programming mistakes inconsistent with the program’s intended design”
Court’s Construction	[Does not require construction]