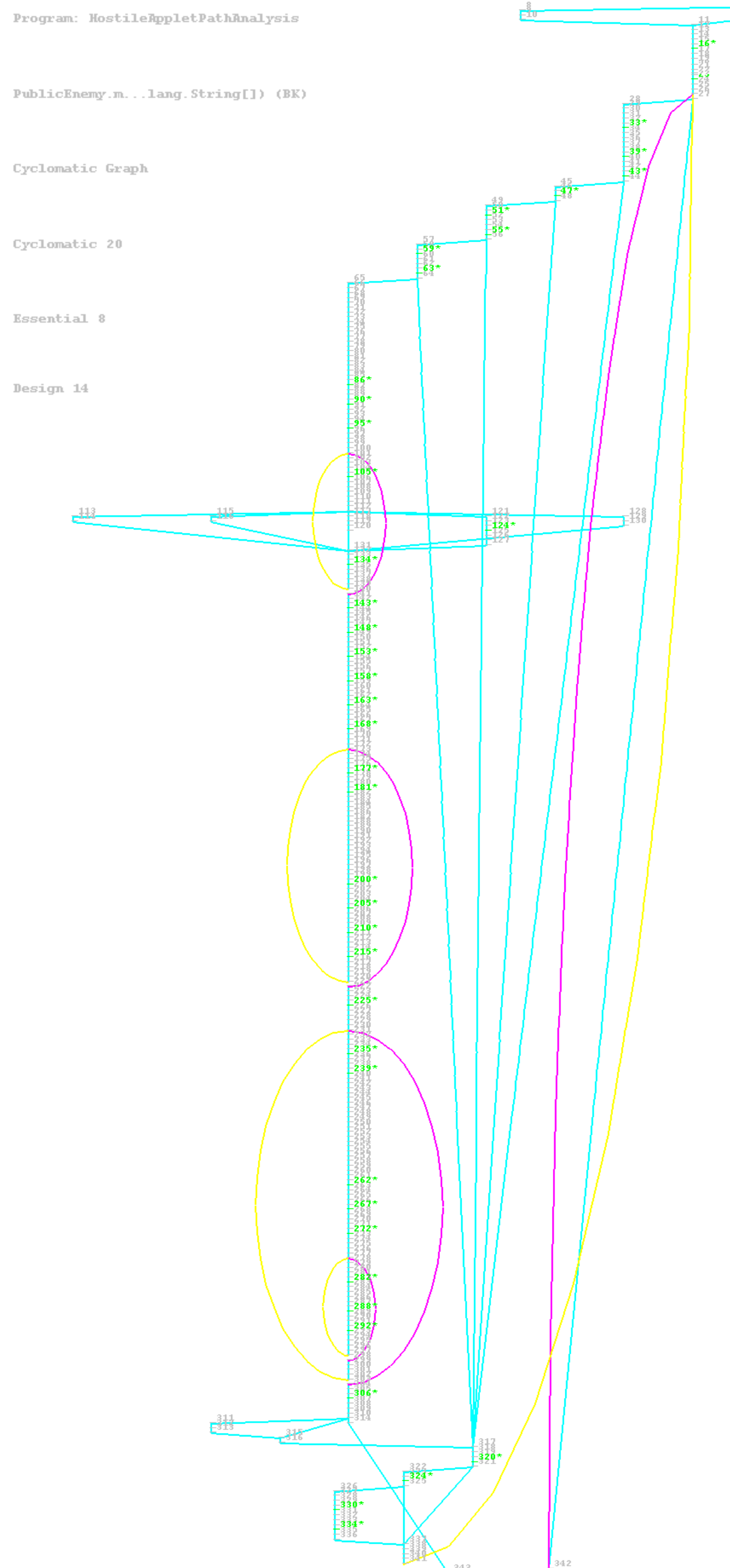
Complexity Analysis of Hostile Applets

Forensics: Using Path-Oriented Metric Analysis to Unravel Hostile Applet Algorithm Patterns, Signatures, Similarities, Authors and Derivations

Abstract

This paper uses known hostile Java applets as an example baseline that could be analyzed and profiled using path analysis to better understand the algorithms, identify their patterns and leverage the analysis to identify signatures, similarities, authors and derivations.



About Hostile Applets

The tool of choice for many hackers has been mobile code. By enticing users to a malicious web page that contains a specially crafted Java applet or application a skilled hacker can wreak havoc.

On the internet there is a comprehensive collection of hostile Java applets. A hostile applet is any applet which, when downloaded, attempts to monopolize or exploit your system's resources in an inappropriate manner. Any applet which performs, or induces you to perform, an action which you would not otherwise care to perform, should be regarded as hostile.

The power and complexity of the Java language make it extremely likely that security holes will continue to appear in years to come. While the number and nature of security holes in Java might be frightening to some, they come as no surprise to computer security professionals and hackers. Any tool as complex and as powerful as Java, being a product of human ingenuity, is bound to have numerous errors in both concept and implementation.[i]

Applets must obey the security rules that are put into place by the developer, using the Java security model. This security model is not infallible - implementing the security model is complicated and rules are not always followed. Additionally, Java security can be sidestepped and there is always the danger the user's computer, or company's network, can be hijacked.[ii] The most serious malicious applets find a way to circumvent Java's security mechanisms and gain complete control of the host machine. These attack applets depend on exploiting a vulnerability in a Java implementation. Other classes of

malicious applets may disturb the victim without circumventing Java's security mechanisms by behaving in an annoying or disruptive way that is within the behaviors permitted by the policy.

Java has always shown promise by being available anywhere, anytime, on any device and is nowadays also integrating with new software inventions such as web services, XML1, mobile and embedded applications. The amount of code developed using this multi-platform technology is huge and demands high security requirements. Various tools and techniques exist that can help to produce robust, reliable and secure software, however there are probably just as many methods for breaking in to code.

Why Hostile Applet Path Complexity Metrics Analysis?

Design, code, and most recently, requirements metrics have been successfully used for predicting fault-prone modules.[iii] They can also be used to profile and uncover security exploitable bugs.

The Cyclomatic complexity of the control flow of the program can show the characteristic style of a programmer and may suggest the manner in which the code was written. Programmers tend to show repeating patterns in their programs. It is possible to identify ownership of a program by examining source code metrics. Programmers are skillful with a limited set of constructs, mainly those that are well known to them and that allow them to write programs faster and more reliably. It is unrealistic to assume that any programmer can develop programs efficiently and correctly using an unfamiliar programming style. This does not only apply to the structure of the programs, but also to the look and feel of it.[iv] Like naturally-evolved human languages, programming languages allow developers to express certain constructs and ideas in different ways. The differences in the way developers express their ideas can be captured in their programming styles, which in turn can be used for author identification. In the context of programming languages, it has been shown that capturing style in source code can help in determining authorship.[v]

McCabe metrics have been used for years in tools used for software forensics. One of the earliest set of techniques for plagiarism detection in software is the attribute counting techniques which count the level of a certain attribute contained within a piece of code. These systems use a number of metrics such as Halstead's software science metrics and McCabe's Cyclomatic complexity.[vi]

Being cognizant of control and data flow paths within your codebase is crucial to uncover software security vulnerabilities located off the beaten path. Security breaches are often a result of multiple interactions within the software that, on the surface, appear innocent. Criminal attackers can disrupt a system by exercising a specific sequence of interdependent decisions that produces unforeseen and possibly disastrous consequences. Analyzing control flow paths and subtree structures is crucial for both testers and developers to verify control flow integrity and uncover serious security flaws hiding in the code. This analysis may also be used to find patterns, signatures and derivations of exploits. As part of a secure, trustworthy software development process, identifying and exercising paths through the code to ensure that program behavior is correct and expected can help find the nastiest of exploits. Cyclomatic complexity and basis path analysis can be very useful in scrutinizing risky code structures and control and data flows. It can also be used to develop profiles of known exploits.

Cyclomatic Complexity has been mentioned as a possible detection method for particularly nasty bugs. In the paper "The Little Hybrid Web Worm that Could," Billy Hoffman, Lead Researcher at SPI Dynamics, and John Terrill, Co-founder of Enterprise Technology, had this to say: "One possible detection method is to examine the Cyclomatic Complexity or McCabe Complexity of a piece of arbitrary JavaScript code. The overall complexity diagram and number of closed loops should remain almost identical regardless of the number of mutations performed on the code. This follows since our mutations change the syntax of the code but not the underlying functionality then the complexity of that functionality should remain the same." The authors are investigating whether a complexity diagram alone is capable of uniquely identifying web malware."[vii] If Cyclomatic complexity can apply to self-modifying script mutations, it can also be used for manual mutations coded by a hostile programmer, in any language. Control flow structure can be used to characterize functionality regardless of syntax variations.

Many tools used to scan for security vulnerabilities operate based on the source code syntax which can be tweaked to avert detection. Finding exploit derivations is not always easy with these tools. As others have pointed out "When the tool failed to detect a bug, it was for one of two reasons: the absence of security rules specifying the vulnerable function, or the presence of a bug in the static analysis tool. Complex code is more likely to contain complicated code constructs and obscure format string functions, resulting in lower detection rates."[viii] McCabe metrics have also been used to estimate the degree of protection achieved by a suite of software protection techniques[ix]

Sometimes source code from your codebase itself is tweaked by changing, or swapping out existing code into a devious exploitable pattern. Once an Applet is written, it can be reused over and over again. Also, for novice programmers, 'slick' features can be added quite easily using code written by someone else.

Most static analysis tools only offer textual information leaving the security analyst the remaining tasks of understanding and patching the code. Visual information is believed to be helpful when fixing security flaws and using dependence graphs for pattern matching security properties has also been suggested.[x]

For these reasons leveraging McCabe path-oriented analysis can help unravel hostile applet algorithmic patterns, signatures, similarities, authors and derivations.

Examples of Hostile Applet Path Complexity Metrics Analysis

Here are some of the types of hostile applets analyzed for software complexity and used as an example baseline for metric pattern match analysis.

DoMyWork.java

This Java applet makes you try to factor a moderately long integer by trial division, and it reports the results back to its home. Clearly the same could be done for many, many other sorts of calculations. While it performs no hostile actions per se, it does put your workstation to work for somebody else, perhaps a business competitor or someone trying to crack codes. To create an applet that does other sorts of work, you can replace the class GetFactor with another working class and adjust the classes Report and ReportServerSocket accordingly.

Ungreatful.java

This Java Applet tries to convince you that your system is having a security problem and that you must now login. If you do so, your user name and password are sent by the browser to the home of this applet. In any event, the applet then proceeds to the applet proceeds with a denial-of-service attack against you.

Other hostile applets include:

* Hijacker.java (A Java trojan horse that can hijack your compiler)
* PublicEnemy.java (A Java trojan horse that directly hacks bytecode)
* Attacker.java (Attacks Beginner.class and makes it deviant)
* HoseMocha.java (A Java application that defends your classes from Mocha)
* Mutator.java (A Java application that mutates and deletes itself after 5 runs)
* Mutator1.java (Mutator.java updated for Java Version 1.1)

Following is a report of path metrics for the source code for some of these applets.

Page 1 01/26/09

Hostile Applet Path Metrics

Program: HostileAppletPathAnalysis

Module Name	v(G)	ev(G)	iv(G)	gdv(G)
PublicEnemy.main(java.lang.Str	20	8	14	3
HoseMocha.main(java.lang.Strin	15	1	9	2
LoginServerSocket.main(java.la	11	7	9	7
ReportServerSocket.main(java.l	11	7	9	7
Login.communicate(java.lang.St	10	1	7	9
Report.communicate(java.lang.S	10	1	7	9
HostileThreads.run()	9	1	4	5
Mutator.main(java.lang.String[	8	1	6	4
Mutator1.main(java.lang.String	8	1	6	4
ScapeGoat.init()	7	4	3	3
Dupe.main(java.lang.String[])	7	1	4	2
GetFactor.GetFactor()	7	4	1	7
OutPanel.action(java.awt.Event	6	1	5	6
Consume.run()	6	4	3	4
DoubleTrouble.run()	6	4	3	2
TripleThreat.run()	6	4	4	3
Forger.run()	5	1	2	2
Attacker.main(java.lang.String	5	1	3	3
NoisyBear.run()	5	3	3	3
HostileThreads.init()	5	1	2	3
Hijacker.main(java.lang.String	5	1	2	1
DoMyWork.init()	4	1	4	4
SilentThreat.run()	4	3	3	2
Ungrateful.run()	4	1	2	3
Wasteful.run()	4	1	2	3
AttackThread.run()	4	3	3	2
DoMyWork.run()	4	1	2	3
Forger.mailMe(java.lang.String	3	1	1	1
Forger.init()	3	1	1	1
ScapeGoat.run()	3	1	2	2
Ungrateful.init()	3	1	3	3
ErrorFrame.ErrorFrame(java.lan	3	1	3	3
Wasteful.fibonacci(long)	3	3	2	3
Ungrateful.start()	2	1	2	2
Wasteful.init()	2	1	2	2
ScapeGoat.start()	2	1	2	2
Wasteful.start()	2	1	2	2
DoubleTrouble.start()	2	1	2	2
Forger.start()	2	1	2	2
Forger.stop()	2	1	2	2
DoMyWork.start()	2	1	2	2
DoubleTrouble.init()	2	1	2	2
TripleThreat.start()	2	1	2	2
NoisyBear.mouseDown(java.awt.E	2	1	2	2
TripleThreat.init()	2	1	2	2
HostileThreads.start()	2	1	2	2
NoisyBear.stop()	2	1	2	2
NoisyBear.start()	2	1	2	2
Consume.init()	2	1	2	2
Consume.start()	2	1	2	2
Login.Login(int)	1	1	1	1
HostileThreads.paint(java.awt.	1	1	1	1
HostileThreads.update(java.awt	1	1	1	1
Calculator.run()	1	1	1	1
Calculator.stop()	1	1	1	0
NoisyBear.init()	1	1	1	1
Consume.stop()	1	1	1	0
HostileThreads.stop()	1	1	1	0
Consume.update(java.awt.Graphi	1	1	1	1
NoisyBear.update(java.awt.Grap	1	1	1	1
NoisyBear.paint(java.awt.Graph	1	1	1	1
Consume.paint(java.awt.Graphic	1	1	1	0
AttackThread.paint(java.awt.Gr	1	1	1	0
Report.Report(java.lang.String	1	1	1	1
AttackThread.update(java.awt.G	1	1	1	1
DoMyWork.stop()	1	1	1	0
AttackThread.stop()	1	1	1	0
OutPanel.OutPanel(java.lang.St	1	1	1	1
WarningPanel.WarningPanel(java	1	1	1	1
SilentThreat.init()	1	1	1	0
SilentThreat.stop()	1	1	1	0
ErrorPanel.ErrorPanel(java.lan	1	1	1	1
SilentThreat.update(java.awt.G	1	1	1	1
SilentThreat.paint(java.awt.Gr	1	1	1	0
SilentFrame.SilentFrame(java.l	1	1	1	1
DoMyWork.paint(java.awt.Graphi	1	1	1	1
ErrorMessage.run()	1	1	1	1
TripleThreat.stop()	1	1	1	0
AttackThread.init()	1	1	1	1
TripleThreat.update(java.awt.G	1	1	1	1
TripleThreat.paint(java.awt.Gr	1	1	1	1
TripleFrame.TripleFrame(java.l	1	1	1	1
ErrorMessage.stop()	1	1	1	0
DoubleFrame.DoubleFrame(java.l	1	1	1	1
Ungrateful.stop()	1	1	1	0
DoubleTrouble.paint(java.awt.G	1	1	1	1
Ungrateful.update(java.awt.Gra	1	1	1	1
Ungrateful.paint(java.awt.Grap	1	1	1	1
DoubleTrouble.update(java.awt.	1	1	1	1
DoMyWork.update(java.awt.Graph	1	1	1	1
Wasteful.stop()	1	1	1	0
DoubleTrouble.stop()	1	1	1	0
Wasteful.update(java.awt.Graph	1	1	1	1
Wasteful.paint(java.awt.Graphi	1	1	1	0
AttackFrame.AttackFrame(java.l	1	1	1	1
Total:	293	137	213	182
Average:	3.08	1.44	2.24	1.92

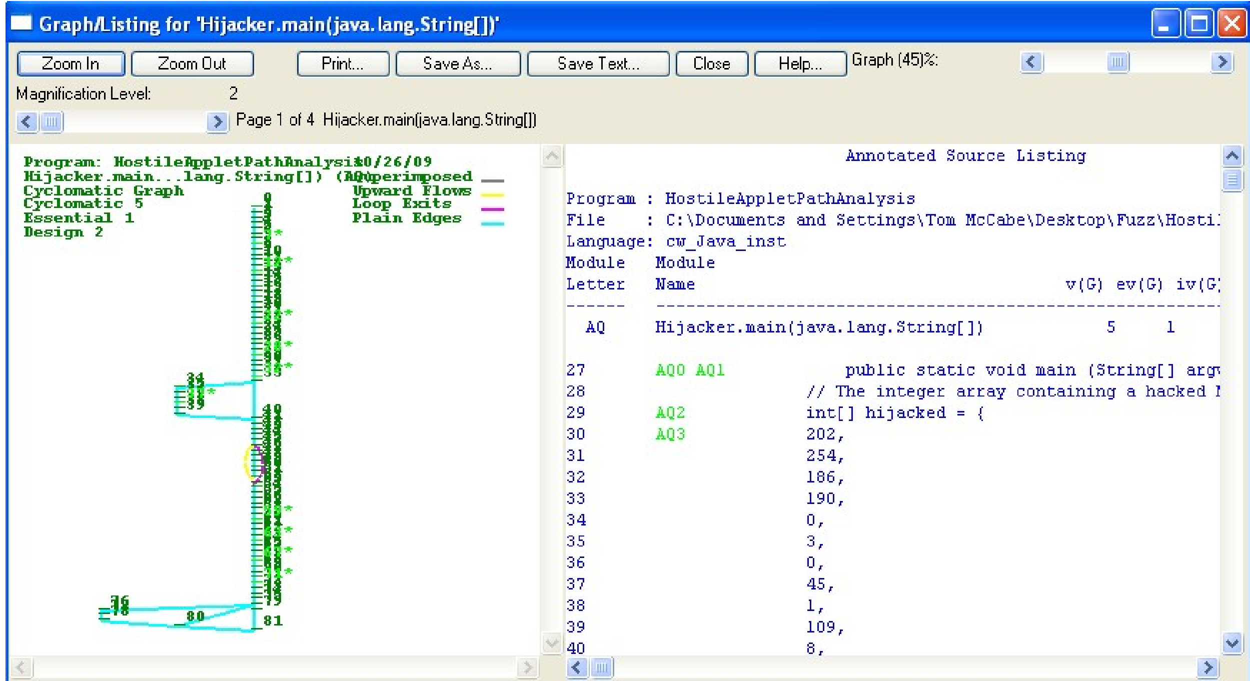
McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

An interesting example of a distinct path complexity pattern is PublicEnemy.java. Given a target directory, this Java application searches it and all of its subdirectories for Java class files. Once a class file is located, PublicEnemy alters the contents of its access_flags for the class, its fields, and its methods. The results are the following:

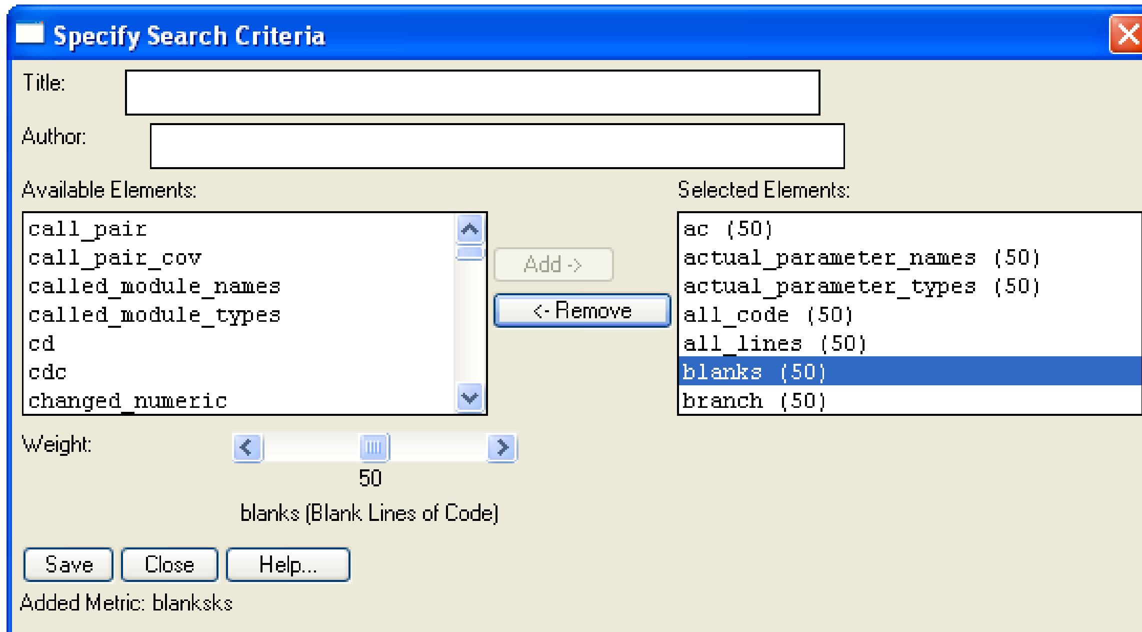
1. The class becomes public.
2. Any final fields and methods become non-final; any non-public fields and methods become public; and all public fields and methods remain so. The following is the code's global data control flowgraph.

Hijacker.java is a Java Trojan that subverts Sun's javac by adding a hostile main class to the user's CLASSPATH ahead of classes.zip. In this case the subverted compiler simply announces its presence and appends the string "Hijacked!" to class files that it produces, but it could just as easily infect them with a Java Platform virus. Again this is another example of a distinct control & data flow pattern. Following is the code's flowgraph.



1. McCabe IQ Source Code Metric Comparisons

Known Hostile Java Applets can be used as a baseline to be profiled. After the analysis is done and saved using the McCabe IQ toolset, it is fairly trivial to search within your source code to find code patterns and potential exploitable logic tweaks. Finding exact matches of patterns can be harder with lower complexity values. In addition to using McCabe path-oriented diagnostics such as Cyclomatic, essential, module design and global data complexity metrics, you may need to experiment using different elements. Path-oriented analysis and profiles can be used to identify exploits, authors and derivations. The analysis and profiles may also be leveraged to determine where in your codebase source code and logic can potentially be tweaked to create an exploit.



```
Date: 11/03/09
                         Module Comparison Tool Report

Program: HostileApplets2
Search Criteria: Control Structure Comparison
Search Domain:
          Program Name  Location Date       Title
          ------------  -------- ----       -----
          HostileApplet (pcf)

Maximum similar modules: 10
Minimum similarity score: 75%
Minimum v(G) for search: 3
Total matches found: 2
Module                                Similar Module(s)                     Similarity Score
------------------------------------- ------------------------------------- ----------------
DoMyWork.init()                       (no matches in threshold)
DoMyWork.paint(java.awt.Graphics)     (no matches in threshold)
DoMyWork.run()                        Ungrateful.run() (pcf HostileApplets2) 100%
DoMyWork.start()                      (no matches in threshold)
DoMyWork.stop()                       (no matches in threshold)
DoMyWork.update(java.awt.Graphics)    (no matches in threshold)
Ungrateful.init()                     (no matches in threshold)
Ungrateful.paint(java.awt.Graphics)   (no matches in threshold)
Ungrateful.run()                      DoMyWork.run() (pcf HostileApplets2)   100%
Ungrateful.start()                    (no matches in threshold)
Ungrateful.stop()                     (no matches in threshold)
Ungrateful.update(java.awt.Graphics)  (no matches in threshold)
```

2. Finding Control and Data Flow Similarities

A good example that can be used to illustrate a derivation can be found within the Hostile Applet baseline. Two close matches within the Hostile Java Applet codebase itself exist. Class metrics for the DoMyWork and Ungrateful Applets look similar and warrant further analysis.

Class Metric Reports for DoMyWork and Ungrateful.Applets:

```
Class Summary for DoMyWork:
  Number of Children:                 1
  Depth in Class Hierarchy:           2
  Response for Class:                 6
  Weighted Methods Per Class:         6
  Coupling of Class:                  0
  Number of Parents:                  2
  Percentage of Public/Protected Data: 100
  Lack of Cohesion of Methods:        78
  Accesses to Public Data:            18
  Depends Upon Child:                 No

  Sum v(G):  13
  Avg v(G):  2.17
  Max v(G):  4
  Max ev(G): 1

Class Member List:
  DoMyWork.paint(java.awt.Graphics)
  DoMyWork.update(java.awt.Graphics)
  DoMyWork.run()
  DoMyWork.stop()
  DoMyWork.start()
  DoMyWork.init()

  Number of non-library modules: 6

Has Specification/Notes? No
```

```
Class Summary for Ungrateful:
  Number of Children:                 1
  Depth in Class Hierarchy:           2
  Response for Class:                 6
  Weighted Methods Per Class:         6
  Coupling of Class:                  0
  Number of Parents:                  2
  Percentage of Public/Protected Data: 100
  Lack of Cohesion of Methods:        76
  Accesses to Public Data:            15
  Depends Upon Child:                 No

  Sum v(G):  12
  Avg v(G):  2.00
  Max v(G):  4
  Max ev(G): 1

Class Member List:
  Ungrateful.paint(java.awt.Graphics)
  Ungrateful.update(java.awt.Graphics)
  Ungrateful.run()
  Ungrateful.stop()
  Ungrateful.start()
  Ungrateful.init()

  Number of non-library modules: 6

Has Specification/Notes? No
```

Here are the flowgraphs and comparison report for the Ungreatful.run module within the Ungrateful Applet and the DoMyWork.run module within the DoMyWork Applet. The path metrics and flowgraphs are strikingly similar. Upon further analysis, the path report helps isolate were the rogue code was tweaked to make a new exploit derivation.

```
Program: HostileAppletPathAnalysis01/02/09
DoMyWork.run() (T)
Cyclomatic Graph
Cyclomatic 4
Essential 1
Design 2
Superimposed
Upward Flows
Loop Exits
Plain Edges
```

```
                              Annotated Source Listing

Program : HostileAppletPathAnalysis
File    : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                          v(G) ev(G) iv(G
------   ---------------------------------------------------------------
   T     DoMyWork.run()                                   4     1

82       T0 T1              public void run() {
83
84                       //  Let the applet tell its lie
85       T2 T3 T4* T5             repaint();
86
87                       //  Let the applet sleep for a while to av
88       T6 T7 T8 T9 T10* T11 T12 T13 T14 T18 T19
                                  try {sleeper.sleep(delay);}
89       T15 T16 T17              catch(InterruptedException e) {}
90
91       T20 T21                  if (controller == null) {
92       T22 T23 T24 T25 T26* T27
                                  Calculator calc = new Calculator(
93       T28 T29 T30 T31 T32* T33
```

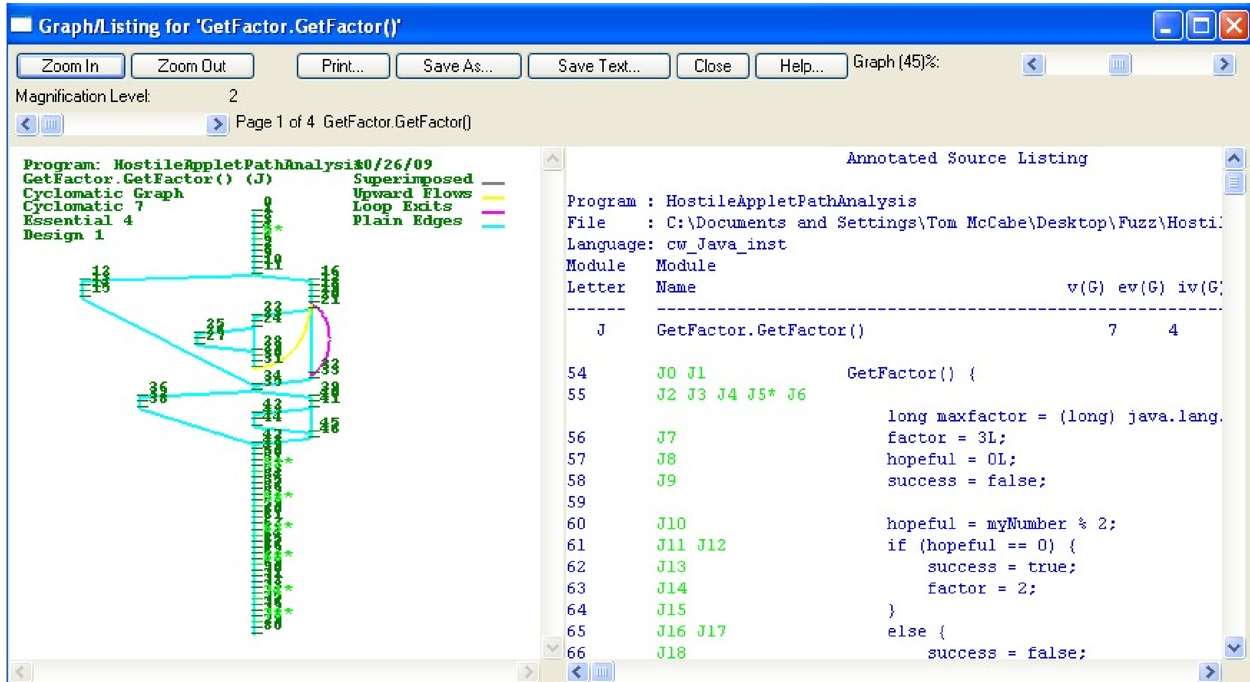
```
Program: HostileAppletPathAnalysis01/02/09
Ungrateful.run() (CH)
Cyclomatic Graph
Cyclomatic 4
Essential 1
Design 2
Superimposed
Upward Flows
Loop Exits
Plain Edges
```

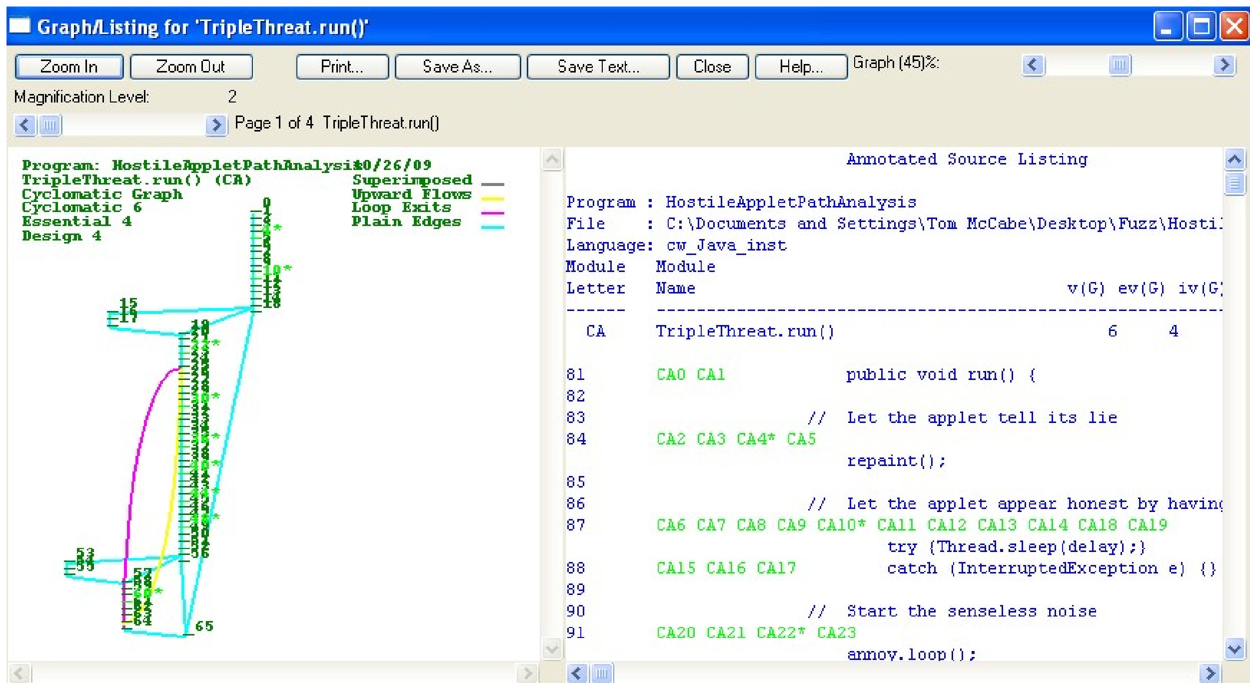
```
                              Annotated Source Listing

Program : HostileAppletPathAnalysis
File    : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                          v(G) ev(G) iv(G
------   ---------------------------------------------------------------
  CH     Ungrateful.run()                                 4     1

67       CH0 CH1            public void run() {
68
69                       //  Let the applet tell its lie
70       CH2 CH3 CH4* CH5
                                  repaint();
71
72                       //  Let the applet sleep for a while to av
73       CH6 CH7 CH8 CH9 CH10* CH11 CH12 CH13 CH14 CH18 CH19
                                  try {sleeper.sleep(delay);}
74       CH15 CH16 CH17           catch(InterruptedException e) {}
75
76       CH20 CH21                if (controller == null) {
77       CH22 CH23 CH24 CH25 CH26* CH27
                                  ErrorMessage err = new ErrorMessag
```

```
+============================================================================+
|                            Compare Paths Report                            |
+============================================================================+

+============================================================================+
|FILE:  C:\Documents and Settings\Tom  |FILE:  C:\Documents and Settings\Tom |
|       McCabe\Desktop\Fuzz\HostileA~  |       McCabe\Desktop\Fuzz\HostileA~ |
|       pplets\Ungrateful.java         |       pplets\DoMyWork.java          |
|MODULE:Ungrateful.run()               |MODULE:DoMyWork.run()                |
|============================================================================|
|Ungrateful.run()                  | 001 |DoMyWork.run()                     |
|----------------------------------------------------------------------------|
|   70: java.awt.Component.repaint()  |   85: java.awt.Component.repaint()  |
|                                      |                                     |
|   73: java.lang.Thread.sleep(long)  |   88: java.lang.Thread.sleep(long)  |
|                                      |                                     |
|   73: #EXCEPTION_THROWN# ==> UNHAN~ |   88: #EXCEPTION_THROWN# ==> UNHAN~ |
|       DLED_EXCEPTION                 |       DLED_EXCEPTION                |
|                                      |                                     |
|   82: <return>                       |   97: <return>                      |
|                                      |                                     |
|* MATCH *                             |                                     |
|============================================================================|
|Ungrateful.run()                  | 002 |DoMyWork.run()                     |
|----------------------------------------------------------------------------|
|   70: java.awt.Component.repaint()  |   85: java.awt.Component.repaint()  |
|                                      |                                     |
|   73: java.lang.Thread.sleep(long)  |   88: java.lang.Thread.sleep(long)  |
|                                      |                                     |
|   73: #EXCEPTION_THROWN# ==> java.~ |   88: #EXCEPTION_THROWN# ==> java.~ |
|       lang.InterruptedException      |       lang.InterruptedException     |
|                                      |                                     |
|   76: controller == null ==> FALSE  |   91: controller == null ==> FALSE  |
|                                      |                                     |
|   82: <return>                       |   97: <return>                      |
|                                      |                                     |
|* MATCH *                             |                                     |
|============================================================================|
|Ungrateful.run()                  | 003 |DoMyWork.run()                     |
|----------------------------------------------------------------------------|
|   70: java.awt.Component.repaint()  |   85: java.awt.Component.repaint()  |
|                                      |                                     |
|   73: java.lang.Thread.sleep(long)  |   88: java.lang.Thread.sleep(long)  |
|                                      |                                     |
|   73: #EXCEPTION_THROWN# ==> NONE   |   88: #EXCEPTION_THROWN# ==> NONE   |
|                                      |                                     |
|   76: controller == null ==> FALSE  |   91: controller == null ==> FALSE  |
|                                      |                                     |
|   82: <return>                       |   97: <return>                      |
|                                      |                                     |
|* MATCH *                             |                                     |
|============================================================================|
|Ungrateful.run()                  | 004 |DoMyWork.run()                     |
|----------------------------------------------------------------------------|
|   70: java.awt.Component.repaint()  |   85: java.awt.Component.repaint()  |
|                                      |                                     |
|   73: java.lang.Thread.sleep(long)  |   88: java.lang.Thread.sleep(long)  |
|                                      |                                     |
|   73: #EXCEPTION_THROWN# ==> java.~ |   88: #EXCEPTION_THROWN# ==> java.~ |
|       lang.InterruptedException      |       lang.InterruptedException     |
|                                      |                                     |
|   76: controller == null ==> TRUE   |   91: controller == null ==> TRUE   |
|                                      |                                     |
|   77: ErrorMessage                   |   92: Calculator                    |
|                                      |                                     |
|*** MISMATCH ***                      |                                     |
+============================================================================+

+============================================================================+
|                                  ANALYSIS                                  |
|----------------------------------------------------------------------------|
|                           Number of basis paths: 4                         |
|                               Number of matches: 3                         |
|                            Number of mismatches: 1                         |
|                Quantitative level of commonality: 0.75                     |
|                 Qualitative level of commonality: HIGH                     |
|----------------------------------------------------------------------------|
| Options: NO DESIGN REDUCTION                                               |
+----------------------------------------------------------------------------+
```

Examples of Hostile Java Applet Algorithm Path Signatures

Following are flowgraphs depicting the path signatures for a number of hostile Java applets.





Graph/Listing for 'Consume.run()'

Zoom In | Zoom Out | Print... | Save As... | Save Text... | Close | Help... | Graph (45)%:

Magnification Level: 2

Page 1 of 4 Consume.run()

```
Program: HostileAppletPathAnalysis10/26/09
Consume.run() (N)              Superimposed
Cyclomatic Graph               Upward Flows
Cyclomatic 6                   Loop Exits
Essential 4                    Plain Edges
Design 3
```

```
                     Annotated Source Listing

Program : HostileAppletPathAnalysis
File     : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                        v(G) ev(G) iv(G
------   ------------------------------------------------------------
  N      Consume.run()                                  6     4

79       N0 N1              public void run() {
80       N2 N3 N4 N5 N6* N7 N8 N9 N10 N14 N15
                                try {Thread.sleep(delay);}
81       N11 N12 N13            catch (InterruptedException e) {}
82       N16 N17                while (n >= 0) {
83       N18 N19 N20 N21 N22* N23 N24 N25 N26 N30 N31
                                try { holdBigNumbers.append(0x7fff
84       N27 N28 N29            catch (OutOfMemoryError o) {}
85       N32 N33 N34* N35
                                repaint();
86       N36 N37                n++;
87       N38 N39 N40            }
88       N41                }
```

Graph/Listing for 'NoisyBear.run()'

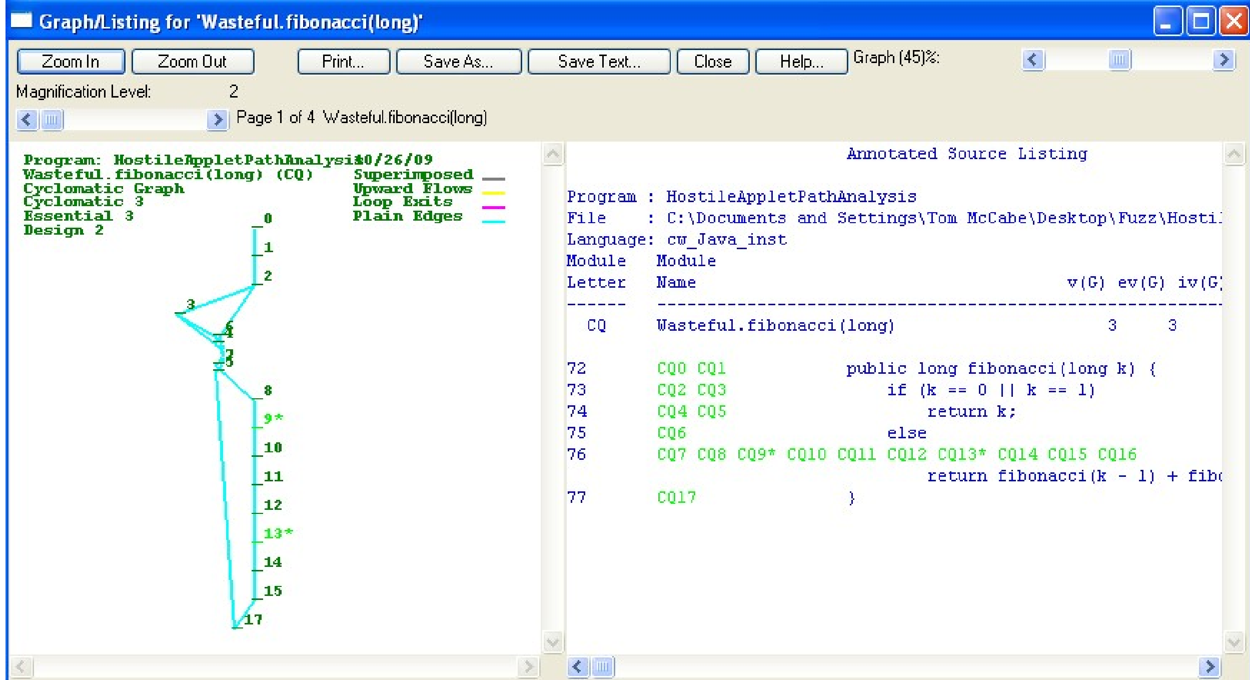
Zoom In | Zoom Out | Print... | Save As... | Save Text... | Close | Help... | Graph (45)%:

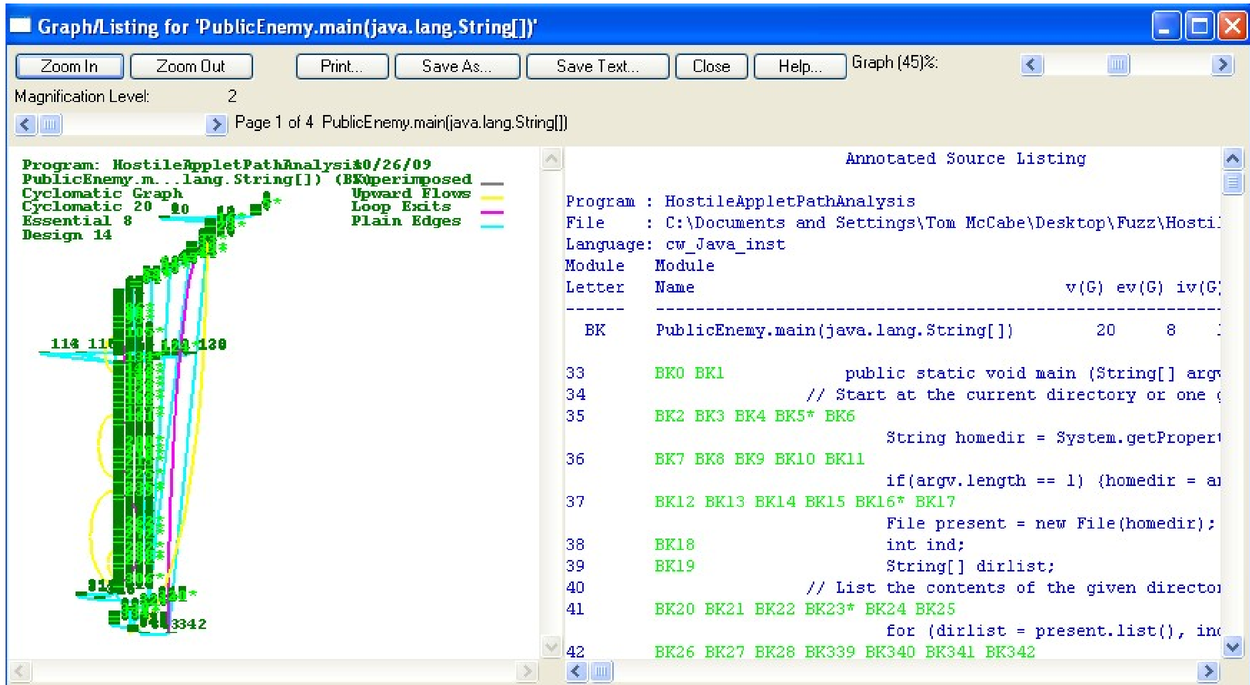
Magnification Level: 2

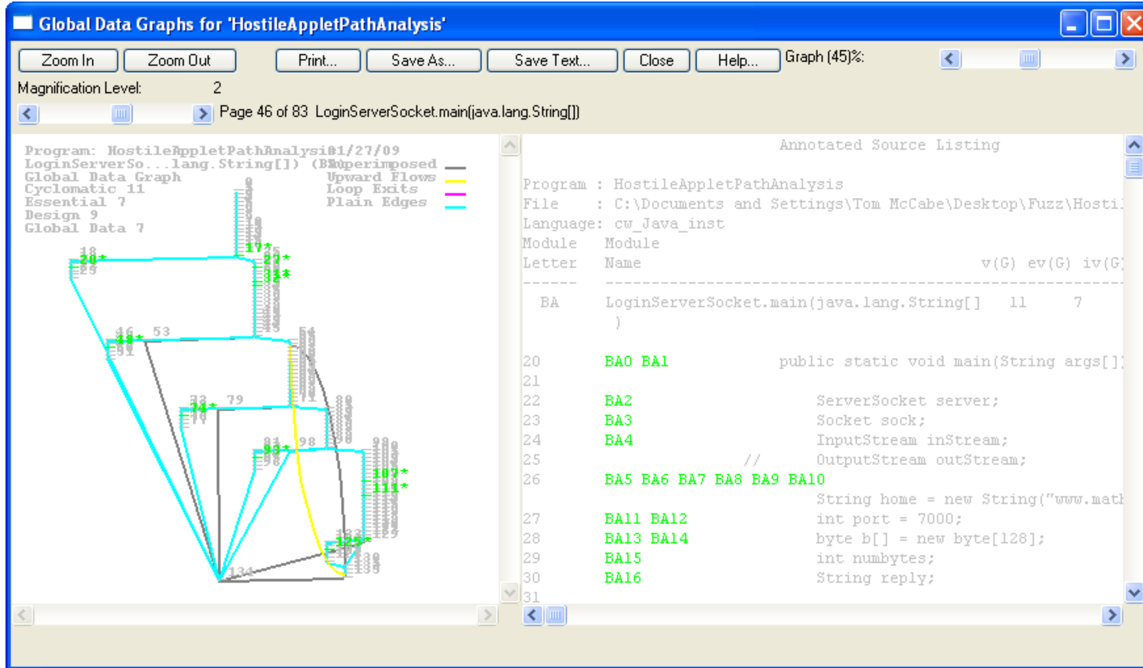
Page 1 of 4 NoisyBear.run()

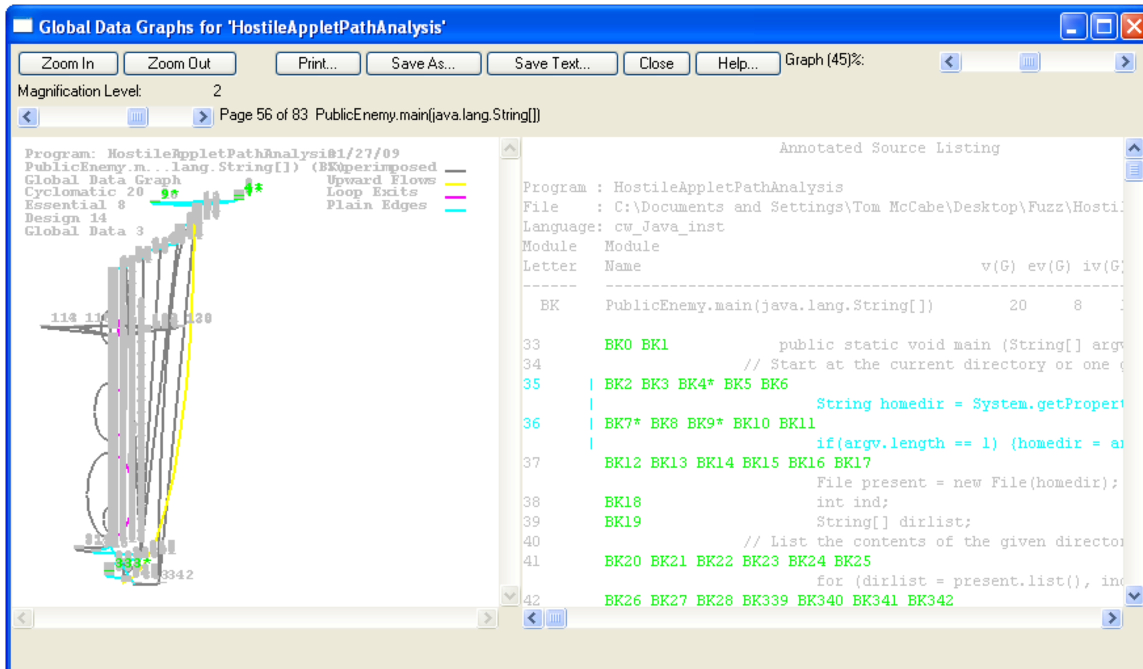
```
Program: HostileAppletPathAnalysis10/26/09
NoisyBear.run() (BG)           Superimposed
Cyclomatic Graph               Upward Flows
Cyclomatic 5                   Loop Exits
Essential 3                    Plain Edges
Design 3
```

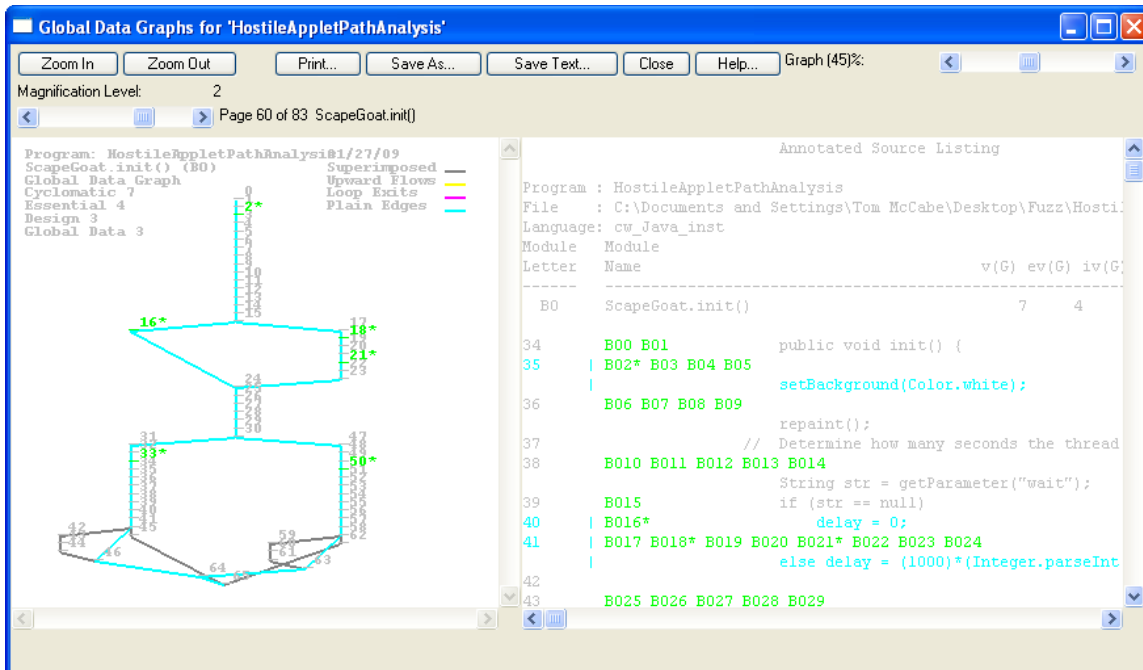
```
                     Annotated Source Listing

Program : HostileAppletPathAnalysis
File     : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                        v(G) ev(G) iv(G
------   ------------------------------------------------------------
 BG      NoisyBear.run()                                5     3

50       BG0 BG1            public void run() {
51       BG2 BG3 BG4 BG5* BG6 BG7
                                if (annoy != null) annoy.loop();
52       BG8 BG9                while (true) {
53       BG10 BG11 BG12 BG13 BG14* BG15
                                rightNow = new Date();
54       BG16 BG17 BG18* BG19
                                repaint();
55       BG20 BG21 BG22 BG23 BG24* BG25 BG26 BG27 BG28 BG32 BG33
                                try { Thread.sleep(1000); }
56       BG29 BG30 BG31         catch (InterruptedException e) {}
57       BG34 BG35 BG36         }
58       BG37               }
```

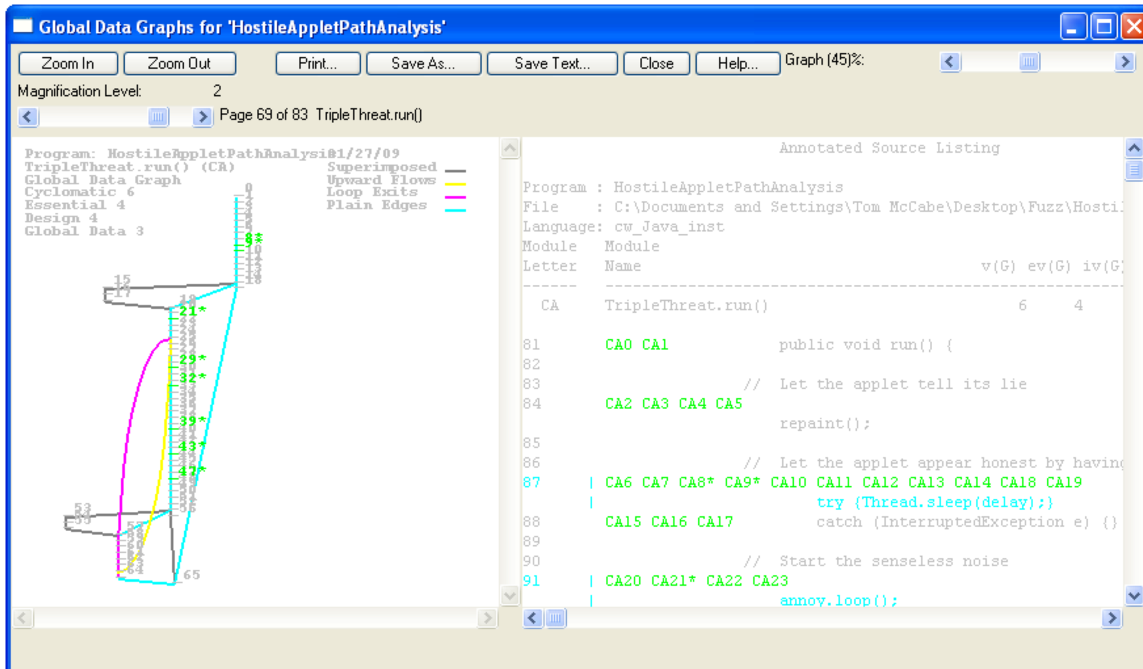

Graph/Listing for 'Wasteful.fibonacci(long)'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 1 of 4 Wasteful.fibonacci(long)
Program: HostileAppletPathAnalysis 10/26/09
Wasteful.fibonacci(long) (CQ)
Cyclomatic Graph
Cyclomatic 3
Essential 3
Design 2
Superimposed
Upward Flows
Loop Exits
Plain Edges
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Module
Letter Name v(G) ev(G) iv(G
CQ Wasteful.fibonacci(long) 3 3
72 CQ0 CQ1 public long fibonacci(long k) {
73 CQ2 CQ3 if (k == 0 || k == 1)
74 CQ4 CQ5 return k;
75 CQ6 else
76 CQ7 CQ8 CQ9* CQ10 CQ11 CQ12 CQ13* CQ14 CQ15 CQ16
return fibonacci(k - 1) + fib
77 CQ17 }


Graph/Listing for 'PublicEnemy.main(java.lang.String[])'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 1 of 4 PublicEnemy.main(java.lang.String[])
Program: HostileAppletPathAnalysis 10/26/09
PublicEnemy.m...lang.String[]) (BK)
Superimposed
Cyclomatic Graph
Cyclomatic 20
Essential 8
Design 14
Upward Flows
Loop Exits
Plain Edges
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Module
Letter Name v(G) ev(G) iv(G
BK PublicEnemy.main(java.lang.String[]) 20 8
33 BK0 BK1 public static void main (String[] arg
34 // Start at the current directory or one
35 BK2 BK3 BK4 BK5* BK6
String homedir = System.getPropert
36 BK7 BK8 BK9 BK10 BK11
if(argv.length == 1) {homedir = a
37 BK12 BK13 BK14 BK15 BK16* BK17
File present = new File(homedir);
38 BK18 int ind;
39 BK19 String[] dirlist;
40 // List the contents of the given directo
41 BK20 BK21 BK22 BK23* BK24 BK25
for (dirlist = present.list(), in
42 BK26 BK27 BK28 BK339 BK340 BK341 BK342


Global Data Graphs for 'HostileAppletPathAnalysis'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 46 of 83 LoginServerSocket.main(java.lang.String[])
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Module
Letter Name v(G) ev(G) iv(G)
BA LoginServerSocket.main(java.lang.String[] 11 7
)
20 BA0 BA1 public static void main(String args[])
21
22 BA2 ServerSocket server;
23 BA3 Socket sock;
24 BA4 InputStream inStream;
25 // OutputStream outStream;
26 BA5 BA6 BA7 BA8 BA9 BA10
String home = new String("www.math
27 BA11 BA12 int port = 7000;
28 BA13 BA14 byte b[] = new byte[128];
29 BA15 int numbytes;
30 BA16 String reply;
31


Global Data Graphs for 'HostileAppletPathAnalysis'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 56 of 83 PublicEnemy.main(java.lang.String[])
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Module
Letter Name v(G) ev(G) iv(G)
BK PublicEnemy.main(java.lang.String[]) 20 8
33 BK0 BK1 public static void main (String[] argv
34 // Start at the current directory or one
35 | BK2 BK3 BK4* BK5 BK6
| String homedir = System.getPropert
36 | BK7* BK8 BK9* BK10 BK11
| if(argv.length == 1) {homedir = ar
37 BK12 BK13 BK14 BK15 BK16 BK17
File present = new File(homedir);
38 BK18 int ind;
39 BK19 String[] dirlist;
40 // List the contents of the given director
41 BK20 BK21 BK22 BK23 BK24 BK25
for (dirlist = present.list(), ind
42 BK26 BK27 BK28 BK339 BK340 BK341 BK342


Global Data Graphs for 'HostileAppletPathAnalysis'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 60 of 83 ScapeGoat.init()
Program: HostileAppletPathAnalysis 01/27/09
ScapeGoat.init() (B0)
Global Data Graph
Cyclomatic 7
Essential 4
Design 3
Global Data 3
Superimposed
Upward Flows
Loop Exits
Plain Edges
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Letter
Module Name
v(G) ev(G) iv(G
B0 ScapeGoat.init() 7 4
34 B00 B01 public void init() {
35 | B02* B03 B04 B05
setBackground(Color.white);
36 B06 B07 B08 B09
repaint();
37 // Determine how many seconds the thread
38 B010 B011 B012 B013 B014
String str = getParameter("wait");
39 B015 if (str == null)
40 | B016* delay = 0;
41 | B017 B018* B019 B020 B021* B022 B023 B024
else delay = (1000)*(Integer.parseInt
42
43 B025 B026 B027 B028 B029


Global Data Graphs for 'HostileAppletPathAnalysis'
Zoom In
Zoom Out
Print...
Save As...
Save Text...
Close
Help...
Graph (45)%:
Magnification Level: 2
Page 69 of 83 TripleThreat.run()
Program: HostileAppletPathAnalysis 01/27/09
TripleThreat.run() (CA)
Global Data Graph
Cyclomatic 6
Essential 4
Design 4
Global Data 3
Superimposed
Upward Flows
Loop Exits
Plain Edges
Annotated Source Listing
Program : HostileAppletPathAnalysis
File : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module Letter
Module Name
v(G) ev(G) iv(G
CA TripleThreat.run() 6 4
81 CA0 CA1 public void run() {
82
83 // Let the applet tell its lie
84 CA2 CA3 CA4 CA5
repaint();
85
86 // Let the applet appear honest by havin
87 | CA6 CA7 CA8* CA9* CA10 CA11 CA12 CA13 CA14 CA18 CA19
try {Thread.sleep(delay);}
88 CA15 CA16 CA17 catch (InterruptedException e) {}
89
90 // Start the senseless noise
91 | CA20 CA21* CA22 CA23
annoy.loop();

Graph/Listing for 'Login.communicate(java.lang.String,java.lang.String)'

Zoom In | Zoom Out | Print... | Save As... | Save Text... | Close | Help... | Graph (45)%:

Magnification Level: 2

Page 1 of 4 Login.communicate(java....String,java.lang.String)

Program: HostileAppletPathAnalysis 10/26/09
Login.communi...a.lang.String) (AZ) Superimposed
Cyclomatic Graph Upward Flows
Cyclomatic 10 Loop Exits
Essential 1 Plain Edges
Design 7

```
                              Annotated Source Listing

Program : HostileAppletPathAnalysis
File    : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                         v(G) ev(G) iv(G
------   ----------------------------------------------------------
  AZ     Login.communicate(java.lang.String,java.l   10     1
          ang.String)

31       AZ0 AZ1            public void communicate (String user,
32       AZ2 AZ3                Socket sock = null;
33                      //      InputStream inStream;
34       AZ4 AZ5                OutputStream outStream = null;
35       AZ6 AZ7                byte b[] = new byte[128];
36       AZ8                    int numbytes;
37       AZ9                    String reply;
38       AZ10 AZ11 AZ12 AZ13 AZ14* AZ15
                                StringBuffer sb = new StringBuffe
39       AZ16 AZ17              InetAddress inaddress = null;
40
41                      //      System.out.println("I'm up to no
42       AZ18 AZ19 AZ33 AZ43 AZ44
```

Graph/Listing for 'Report.communicate(java.lang.String,java.lang.String)'

Zoom In | Zoom Out | Print... | Save As... | Save Text... | Close | Help... | Graph (45)%:

Magnification Level: 2

Page 1 of 4 Report.communicate(java...String,java.lang.String)

Program: HostileAppletPathAnalysis 10/26/09
Report.commun...a.lang.String) (BM) Superimposed
Cyclomatic Graph Upward Flows
Cyclomatic 10 Loop Exits
Essential 1 Plain Edges
Design 7

```
                              Annotated Source Listing

Program : HostileAppletPathAnalysis
File    : C:\Documents and Settings\Tom McCabe\Desktop\Fuzz\Hosti
Language: cw_Java_inst
Module   Module
Letter   Name                                         v(G) ev(G) iv(G
------   ----------------------------------------------------------
  BM     Report.communicate(java.lang.String,java.   10     1
          lang.String)

34       BM0 BM1            public void communicate(String teststr
35       BM2 BM3                Socket socker = null;
36       BM4 BM5                OutputStream outerStream = null;
37       BM6 BM7                byte by[] = new byte[4096];
38       BM8                    int numberbytes;
39       BM9 BM10               InetAddress inneraddress = null;
40       BM11 BM12              String response = null;
41       BM13 BM14 BM15 BM16 BM17* BM18
                                StringBuffer responsebuf = new Str
42                      //      System.out.println("I'm up to no
43       BM19 BM20 BM34 BM44 BM45
                                try {
44       BM21 BM22 BM23 BM24 BM25* BM26
```

[i] Java Insecurity, Mark D. LaDue, 1996

[ii] Plain English: Risks of Java Applets And Microsoft ActiveX Controls, Jennifer M. Marek
GIAC Security Essentials Certification (VER 1.3), 4 March 2002, SANS Institute

iii "Comparing Design and Code Metrics for Software Quality Prediction" Yue Jiang, Bojan Cukic, Tim Menzies, Nick Bartlow, The Lane Department of Computer Science and Electrical Engineering, WVU

iv "Authorship Analysis: Identifying The Author of a Program" Ivan Krsul Eugene H. Spafford, The COAST Project - Department of Computer Sciences, Purdue University

v Identifying Source Code Authorship, Robert Lange, Jonathan Max-Sohmer, Maxim Shevertalov, Jay Kothari, Spiros Mancoridis, Software Engineering Research Group, Department of Computer Science

vi "SOURCE CODE AUTHORSHIP ANALYSIS FOR SUPPORTING THE CYBERCRIME INVESTIGATION PROCESS" Georgia Frantzeskou, Stefanos Gritzalis Laboratory of Information and Communication Systems Security, Aegean University, Stephen G. MacDonell, School of Computer and Information Sciences Auckland University of Technology

vii " The Little Hybrid Web Worm that Could" Billy Hoffman, SPI Dynamics, John Terrill Co-founder Enterprise Management Technology

viii "Measuring the Effect of Code Complexity on Static Analysis Results" James Walden, Adam Messer and Alex Kuhl, Department of Computer Science Northern Kentucky University

ix "Realistic and Affordable Cyberware Opponents for the Information Warfare BattleSpace"
Martin R. Stytz, Ph.D., Sheila B. Banks, Ph.D., Michael J. Young, Ph.D., Air Force Research Laboratory
Wright-Patterson AFB, OH 45431

x "Pattern Matching Security Properties of Code using Dependence Graphs" John Wilander and Pia Fak, Dept. of Computer and Information Science, Linkopings Universitet

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920