A Formalisation of Digital Forensic String Search Specifications

Benjamin Aziz
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

Abstract

Computer forensic tools are tools concerned with the identification, recovery, preservation and analysis of evidence in digital media involved in criminal investigations and illegal cases. String search tools are a special class of such forensic tools, which are used to search for specific queries in the investigated medium. However, one main issue we find in literature mainly is that computer forensic tools, including string search tools, are generally specified in an informal and oftentimes ambiguous and inconstant manner. We adopt in this short paper formal specification approaches, in particular Event-B, in order to define clearly the behaviour and properties of string search tools, a class of computer forensic tools, and investigate whether there are any requirements that cannot be achieved. Our results show that some basic and optional requirements as stated in literature related to string search tools could be not be consistent and some are unnecessarily restrictive.

Keywords: Digital Forensics, Event-B, Formal Methods, String Search Tools.

1 Introduction

The term *computer forensics tools* refers to all software and hardware tools used to identify, preserve, recover, analyse and present facts and opinions about information and evidence recovered from computers involved in criminal and illegal cases [5, 6]. The National Institute of Standards and Technology (NIST) runs a project on Computer Forensic Tool Testing (CFTT) [14], which aims at raising the assurance of computer forensic tools by providing informal definitions of the various computer forensic tools and their requirements. These definitions and requirements are then used to derive functional specifications, test procedures, criteria, sets and hardware. We take this assurance process here to another level where the functional specifications and some of the properties of the computer forensic tools are formally defined and verified using well-established formal modelling languages, in this case, the Event-B formal method [2]. Such an approach fulfills the need for the integration of more robust and rigorous scientific methods into the area of digital forensic tools development [12, 6, 5], much needed as a result of the critical nature of digital forensics investigations.

The application of formal modelling and analysis techniques to digital forensics is by no means a new idea, though it has been under-researched in many aspects. In [13], the B method [1] is used for developing incosistency checks and verifying the correctness of digital evidence. The B method has also been used to formally specify and refine write blocker systems in [10, 9] based on NIST's informal definitions of these systems in [15] and provide formal definitions of the properties of these systems.

This short paper presents the initial results of formally modelling Digital Forensic String Search (DFSS) tools. DFSS tools are essential in computer forensics investigations, since they allow the investigator to search for a relevant query (e.g. a string of characters referring to a weapon) under some criteria and within a digital medium (e.g. a file on the suspect's computer machine), and then display the hit results as evidence. The rest of the paper is organised as follows. In Section 2, we review the

NIST requirements for string search forensics tools. In Section 3, we provide an initial sketch of a formal model of these tools using the formal language of Event-B. In Section 4, we outline the results of this model in terms of how some requirements proved to be inconsistent and others overly restrictive. Finally, in Section 5, we conclude the paper.

2 NIST's DFSS Requirements

We give an overview here of the DFSS requirements that feature in our formal model later. These include two basic requirements and one optional requirement. We also discuss some assumptions we made regarding these requirements arising from the ambiguity of the model description stated in [7, 8].

2.1 Basic Requirements

In [7], two basic requirements for DFSS tools were stated.

SS-BR-01 "The response returned by a query is equal to the match set for the query."

SS-BR-02 "The tool shall search using one or more specified character representations."

2.2 Optional Requirements

In addition to the above two basic requirements, there were 17 optional requirements defined in [7]. We consider here one of these requirements that we found to be interesting (most of the rest are related to either meta characters or choice of search algorithms).

SS-OR-03 "If the tool searches an unordered search arena and a hit count of n is specified then the response is any $k = min(n,m)$ matches, where m is the number of strings in the match set."

2.3 Modelling Assumptions

Before constructing the formal model for a DFSS tool, we had to make some assumptions regarding the requirements mentioned in Sections 2.1 and 2.2 above. We discuss these assumptions next.

For the case of SS-BR-01, there was some ambiguity as to what the meaning of the word *equal* is. We found some clarification in [8] through the definition of the first two test assertions, SS-CA-01 and SS-CA-02, for DFSS tools.

SS-CA-01 "All elements of the response set are members of the match set for the query."

SS-CA-02 "All members of the match set are included in the response set for the query." This implies that the equality statement is related to set inclusion. However, we find a problematic statement in [8, §6]:

"The response set is the set of strings actually returned by a query."

which contradicts with the definition of a response set given in [8, §5] as:

"The set of search hits returned by a query."

where a search hit itself is defined, again in [8, §5], as:

"The string matching the query and a *location description* within the search area."

Therefore, clearly the equality relation definition in terms of set inclusion ignores the fact that a response set consists of pairs of strings and location descriptions, as opposed to the definition of a match set, which consists only of string elements. As a result, our **first assumption** was to assume that the equality relation only applies to the first element of the response pairs (i.e. the string element) and ignores the location description. We interpreted this equality then to mean that every string part of the hits returned by a search algorithm must be an element of the match set, and every element of the match set must appear in at least one hit.

The **second assumption** we made was related to the following statement in [8, §5], part of the definition of what a match set is:

"... the expected result to be returned by executing a query."

which clearly assumes the presence of a hypothetical *ideal* algorithm that would return the "expected result" corresponding to some query, against which the actual results returned by the tool will compared. We had to make the assumption that such an ideal algorithm existed.

Finally, the **third assumption** we made was to conclude that the first item in the definition of a location description (in [7, 8, §5]):

"1. Matching string"

was simply redundant information, since it is the same as the first element in the definition of a search hit "The string matching the query".

3 A Formal Model of DFSS

We used Event-B [2] as the formal language for specifying a DFSS tool, following a similar approach to previous successful modelling attempts presented in [3, 4]. The model is represented by the Event-B machine and its context shown in Figure 1 modelled using the Rodin platform [11]. The model of the context presents two constants; one as an instance of a string search algorithm (`stringsearchalgorithm`), and the other, as an instance of an ideal string matching algorithm (`expectedmatchset`). The context also defines a number of sets, representing the basic types of the different terms of [7, 8, §5]. The types of the two constants are defined by axioms, `axm1` and `axm2`. The difference between the two is that `stringsearchalgorithm` takes also as part of the input the object being searched (e.g. file, document etc.), which is an element of the `SearchUniverse`, and provides additionally as output a location description as a triple consisting of elements of the sets of `ObjectIdentification`, `OffsetWithinObject` and `MatchingStringLength`.

The model of the machine, on the other hand, consists of two events: a machine state initialisation event, which initialises the `stringsearchresult` machine variable to an empty value and the `query` variable to some query value. The `query` is computed against an `object` using the search algorithm, `stringsearchalgorithm`, in the `StringSearch` event, and the result is assigned as the new value to `stringsearchresult`. The new `stringsearchresult` value respects requirement SS-OR-03 on the cardinality of the returned result, through conditional guards `grd2` and `grd3`, which implement that requirement. Finally, `inv3` expresses requirement SS-BR-01, whereas requirement SS-BR-02 we found to be superficial since it constitutes part of the type of `query` expressed by `inv2`.

```
MACHINE
DFSSMachine

SEES
DFSSContext

VARIABLES
stringsearchresult, query

INVARIANTS
inv1: stringsearchresult ∈ ℘(Strings×
(ObjectIdentification × OffsetWithinObject × MatchingStringLength))
inv2: query ∈ ℘(SearchPattern × CharacterRepresentation × IgnoreCase×
TextDirection × SearchType)
inv3: (∀x.x ∈ stringsearchresult ⇔ (prj1(x) ∈ expectedmatchset(query))) ∧ (∀x.x ∈ expectedmatchset(query) ⇔ (∃y.y ∈
stringsearchresult ∧ (x = prj1(y))))

EVENTS

INITIALISATION
act1:  stringsearchresult:= ∅
act2:  query:∈ ℘(SearchPattern×CharacterRepresentation×
            IgnoreCase×TextDirection×SearchType)
END


StringSearch
ANY
object, k

WHERE
grd1: object∈SearchUniverse
grd2: card(stringsearchalgorithm(query↦object))≤
    card(expectedmatchset(query))  ⇔  (k=card(stringsearchalgorithm(query↦object)))
grd3: card(stringsearchalgorithm(query↦object))≥
    card(expectedmatchset(query))  ⇔  (k=card(expectedmatchset(query))

THEN
act1:  stringsearchresult:| (stringsearchresult'=stringsearchalgorithm(query↦object))∧
     (card(stringsearchresult')=k)
END
END
```

```
CONTEXT
DFSSContext

SETS
Strings, SearchUniverse, SearchPattern, CharacterRepresentation, IgnoreCase, TextDirection, SearchType,
ObjectIdentification, OffsetWithinObject, MatchingStringLength

CONSTANTS
stringsearchalgorithm, expectedmatchset

AXIOMS
axm1:   stringsearchalgorithm ∈ (℘(SearchPattern × CharacterRepresentation × IgnoreCase × TextDirection ×
SearchType) × SearchUniverse) → ℘(Strings × (ObjectIdentification × OffsetWithinObject × MatchingStringLength))
axm2: expectedmatchset ∈ ℘(SearchPattern × CharacterRepresentation × IgnoreCase × TextDirection × SearchType) →
℘(Strings)

END
```

Figure 1: An Event-B Formal Model of a DFSS tool

4 Results: On Discharging Proof Obligations

The main result that we obtained was that we were not able to prove `inv3`. This was mainly due to the realisation that requirement SS-BR-01 is inconsistent with the optionanal requirement, SS-OR-03, modelled as part of the behaviour of the `StringSearch` event. This inconsistency is demonstrated through the following counter-example.

Counter-example Assume there are n number of hits when a DFSS tool searches a document based on a query, and that the expected number of matches for that query is m, according to some ideal search algorithm. There are three cases, that this scenario could lead to:

First, that $n < m$, in which case the hit results will necessarily have covered all the values in the expected match set. This assumes that the DFSS tool does not introduce foreign values contrary to what is expected from the ideal search algorithm. As an example, the hits set might be $\{(knife, \ell_1), (gun, \ell_2), (knife, \ell_3)\}$, occurring at three locations in the searched document, ℓ_1, ℓ_2 and ℓ_3 (for simplicity, we ignore the location description), and the expected match set is $\{knife, gun, poison, bomb\}$. Although this case breaks SS-BR-01, since the two sets of strings are not equal, in our opinion, it is a *safe* case, since all the returned hits are *expected matches*, and therefore, it also demonstrates that SS-BR-01 is an unnecessarily strong requirement, due to SS-CA-02 [8].

Second, $m < n$, in which case we cannot conclude anything, simply because this could be due to either the whole of the match set being found repetitively in the hit set, e.g. in the case of the following set $\{(knife, \ell_1), (gun, \ell_2), (poison, \ell_3), (bomb, \ell_4), (poison, \ell_5), (gun, \ell_6)\}$, or in the ase of the subset $\{(knife, \ell_1), (gun, \ell_2), (knife, \ell_3), (gun, \ell_4), (gun, \ell_5), (gun, \ell_6)\}$. The former possibility maintains the requirement SS-BR-01, whereas the latter breaks it. It, furthermore, demonstrates the safety weakness of SS-OR-03, as the returned number of hits will be only m (in our example, 4), which may not cover all the values in the match set given the above two possibilities of hit sets. For example, if we were returning $\{(gun, \ell_2), (poison, \ell_3), (poison, \ell_5), (gun, \ell_6)\}$ from the first hit set, or returning $\{(gun, \ell_2), (gun, \ell_4), (gun, \ell_5), (gun, \ell_6)\}$ from the second hit set, both would satisfy SS-OR-03 but clearly miss crucial forensic evidence, since the original hit sets contain hits with strings that are members of the match set $\{knife, gun, poison, bomb\}$.
Third and finally, $n = m$, in which case, there are no issues in regard to SS-OR-03, since the two sets are equal in size. For example, the hit set could be the set $\{(knife, \ell_1), (gun, \ell_2), (poison, \ell_3), (bomb, \ell_4)\}$, or the hit set could be the set $\{(knife, \ell_1), (gun, \ell_2), (knife, \ell_3), (gun, \ell_4)\}$, neither of which misses any evidence, since the whole hit set must be returned and it is included in the match set. It does however, again, demonstrate that SS-BR-01 is unnecessarily strong because of SS-SC-02, since it may be the case that the whole match set is not found in the hit set, as in the second example.

5 Conclusion

As is often the case with informal specifications, we find that NIST's specifications of DFSS tools as outlined in [7, 8] are at best ambiguous, and often contain inconstant and contradictory requirements and definitions, as we have concluded from the initial results presented in this short paper. These results have demonstrated that requirement SS-BR-01 cannot be proven (case of $n < m$) as it is unnecessarily strong, and additionally, requirement SS-OR-03 is unsafe (case $m < n$). In addition to these formal results, we also found that there were a few ambiguities in the definitions of terms underlying a DFSS tool, that we had to make some assumptions about when defining the formal Event-B model. These were outlined in

Section 2.3. The results are in-line with other research work we have performed, e.g. [4, 3], where we demonstrated similar benefits of using formal methods in strengthening our understanding of the digital forensics tools specifications and properties.

Our main recommendations would be to weaken requirement SS-BR-01 (by removing the overly restrictive assertion SS-CA-02) and remove the superficial requirement SS-BR-02 altogether as it does not add anything new. Additionally, we recommend to strengthen SS-OR-03 such that the returned $k = min(n,m)$ hits satisfies the new definition of SS-BR-01, and it is not just weakly defined as being simply *any k* hits.

It is worth mentioning here that we found other ambiguities that we simply ignored in our initial model, such as, for example, requirement SS-OR-10 [7], which states that "The [DFSS] tool may combine queries with logical operations such as *and*, *or*, or *not*." This requirement clearly leaves ambiguous the definitions of *and*, *or* and *not*, since these cannot be assumed to be logical connectors as the outcome of applying the DFSS tool is a set of strings and location descriptions, and not Booleans, and therefore some definition of the semantics of *and*, *or* and *not* is required.

References

[1] J.-R. Abrial. *The B Book.* Cambridge University Press, 1996.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Design.* Cambridge University Press, 2010.

[3] B. Aziz. Modelling and refinement of forensic data acquisition specifications. *Digital Investigation*, 11(2):90–101, June 2014.

[4] B. Aziz, P. Massonet, and C. Ponsard. A formal model for forensic storage media preparation tools. In *Proc. of the 11th International Conference on Security and Cryptography (SECRYPT'14), Vienna, Austria*, pages 1–6. IEEE, August 2014.

[5] E. Casey. *Digital Evidence and Computer Crime – Forensic Science, Computers and the Internet 3rd Ed.* Elsevier, 2011.

[6] E. Casey and C. Rose. *Forensic Discovery: Handbook of Digital Forensics and Investigation.* Academic Press, 2010.

[7] Forensic String Searching Tool Requirements Specification: Public Draft 1 of Version 1.0, Jan. 2008.

[8] Forensic String Searching Tool Test Assertions and Test Plan, Mar. 2018.

[9] A. Enbacka. Formal methods based approaches to digital forensics. Master's thesis, Åbo Akademi University, 2007.

[10] A. Enbacka and L. Laibinis. Formal specification and refinement of a write blocker system for digital forensics, 2005.

[11] Event-B.org. Rodin Platform and Plug-in Installation . `http://www.event-b.org/` [Online; Accessed on September 1, 2021].

[12] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6:2–11, 2009.

[13] P. Gladyshev and A. Enbacka. Rigorous development of automated inconsistency checks for digital evidence using the b method. *International Journal of Digital Evidence*, 6(2), 2007.

[14] NIST. Computer forensics tool testing (cftt) project web site. `www.cftt.nist.gov/` [Online; Accessed on September 1, 2021].

[15] NIST. Software write block tool specification and test plan (v3.0). Technical report, NIST, 2003.

Author Biography

Benjamin Aziz is a Senior Lecturer at the School of Computing, University of Portsmouth. Benjamin holds PhD degree in formal verification of computer security from Dublin City University (2003) and has research interests and experience in the field of computer and information security, with over 130 publications related to areas such as security engineering of large-scale systems, IoT and SDN security, formal methods, requirements engineering and digital forensics. He is on board several program committees for international conferences and working groups, including ERCIM's FMICS, STM, Cloud Security Alliance and IFIP WG11.3.