2010

Reverse engineering for mobile systems forensics with Ares

John Tuttle
Robert J. Walls
Erik G Learned-Miller, *University of Massachusetts - Amherst*
Brian Neil Levine

Reverse Engineering for Mobile Systems Forensics with Ares

John Tuttle Robert J. Walls Erik Learned-Miller Brian Neil Levine
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{jtuttle,rjwalls,elm,brian}@cs.umass.edu

ABSTRACT

We present Ares*, a reverse engineering technique for assisting in the analysis of data recovered for the investigation of mobile and embedded systems. The focus of investigations into insider activity is most often on the data stored on the insider's computers and digital devices — call logs, email messaging, calendar entries, text messages, and browser history — rather than on the status of the system's security. Ares is novel in that it uses a data-driven approach that incorporates natural language processing techniques to infer the layout of input data that has been created according to some unknown specification. While some other reverse engineering techniques based on instrumentation of executables offer high accuracy, they are hard to apply to proprietary phone architectures. We evaluated the effectiveness of Ares on call logs and contact lists from ten used Nokia cell phones. We created a rule set by manually reverse engineering a single Nokia phone. Without modification to that grammar, Ares parsed most phones' data with 90% of the accuracy of a commercial forensics tool based on manual reverse engineering, and all phones with at least 50% accuracy even though the endianess for one phone changed.*

Categories and Subject Descriptors

D.0 [**Software**]: General; E.2 [**Data**]: Data Storage Representations—*Primitive Data Items*

General Terms

Security

Keywords

Data Recovery, Forensics, Reverse Engineering, Mobile Phones

1. INTRODUCTION

The actions of malicious insiders pose the greatest threat when no software vulnerability is involved. For example, an employee does not require privilege escalation to phone a third party and exchange inside information, and no intrusion detection system will catch the event. Hence, the focus of investigations into insider activity is more often on the data stored on the insider's computers and digital devices — call logs, email messaging, calendar entries, text messages, and browser history — rather than on the status of the system's security. Furthermore, it is critical to retrieve digital artifacts that the user has attempted to delete in order to hide past events.

Cell phones are particularly valuable to investigations of policy violation (including criminal investigations) because they are pervasive and are often carried into the event or crime itself. Call records, saved texts, and address books, as well as other circumstantial evidence, can identify important social relationships for investigators. A phone may also contain direct evidence, such as stolen intellectual property or a record of its transfer. It is insufficient to simply browse through a phone using its own interface to recover data, as deleted information will certainly not be available. Instead, the raw bytes must be analyzed via data recovery tools that understand the proprietary format and are distinct from the device's API.

For many reasons it is a significant challenge to recover meaningful structures from the raw bytes acquired from a cell phone's persistent storage. First, manual reverse engineering (RE) of unknown formats is an extremely time-intensive process; near real-time analysis is impractical for devices that have not been examined previously. Second, unlike desktop file systems, acquisition of the data stored on a phone is an unstandardized process, and it is not possible to recover deleted data by simply using the phone's interface (or backup software). Moreover, using the phone's interface risks modifying the data itself. Third, existing automated RE techniques are not applicable to phone data as almost all *instrument* the software that processes the data. To leverage instrumentation, it is insufficient to have the hardware used or a hardware emulator (unlikely in the case of proprietary chipsets). Instrumentation requires the investigator program an infrastructure that is specialized to the hardware *and* OS of the device, which in the case of phones are protean, closed source, and obfuscated. Instrumentation and taint analysis for Intel-based Windows and Linux systems are formidable tasks [3, 18] that are not applicable to phone systems — with over 190 new cell phones introduced to the US market in the last year alone from about ten major manufacturers (see `www.mobiledia.com`), it would be a herculean effort to adapt

InsiderThreats'10, October 8, 2010, Chicago, Illinois, USA.

such systems to all that are available used or new. (Similar challenges are posed by proprietary medical systems [13].)

In this paper, we propose that reverse engineering embedded systems for data recovery and investigation requires a data-driven approach. Our work is a departure from reverse engineering solutions for Intel/MS desktop systems, which tend to be heavy-weight, relying on instrumented binaries [4, 5, 9], human-readable delimiters [10, 11] or, in the case of commercial forensics tools, manual effort. In contrast, the phone market has varied and proprietary hardware and software, non-delimited machine-oriented output, and an endlessly growing set of platforms. Another advantage of our approach is that it supports (though does not solve) near real-time analysis, as we demonstrate in our evaluation.

In our approach, called the *Adaptable Reverse Engineering System (Ares)*, we examine only the raw bytes of the application data, avoiding the steep challenge of architecting an instrumentation system for every possible phone platform. Ares uses a grammar to encode data formats that are observed in embedded systems. The rules of this grammar are then applied to newly observed systems. We make several extensions to a known parsing algorithm to find the *maximum likelihood parse* for a particular machine data format. We expect the grammar to be extended in an iterative process, much like firewall (such as `snort`) and software assurance rule sets (`Fortify` [6]) are developed and strengthened over time. We created grammar rules that encoded the phone call log format of a 2001 Nokia model 3360 phone. We then applied the rule set to ten other Nokia phones released from 2003–2005. Ares could parse some phone records with over 90% accuracy, with all results above 50% even for phones with a changed endianess. We believe our approach provides an initial step towards addressing the problems that are specific to mobile systems data recovery.

2. ALGORITHM DESIGN

In this section, we present our scenario's assumptions and a formal definition of the problem of reverse engineering of non-delimited data formats. We then propose Ares, a data-driven reverse engineering tool.

Motivation. Forensic investigation of cell phones is difficult for many reasons [2, 15]. To perform a forensically sound data recovery, investigators must first obtain stored data from the phone. This acquisition alone is cumbersome but can often be achieved by using special tools. It is helpful but insufficient to only obtain data from the phone using the phone's interface or software. Deleted data cannot be retrieved and information that has been deleted cannot be recovered. Moreover, the process of viewing data can alter it (e.g., changing last accessed times).

Once acquired, the data must be interpreted in order to extract useful information. However, phone software is largely proprietary and manufacturers will resist helping investigators in order to protect trade secrets. As a result, the data format must be reverse engineered. A few companies sell tools that parse phone data using knowledge gained from manual reverse engineering. However, this process is laborious and must be repeated often as new cell phone models are released regularly. As a consequence, these tools can be very expensive — some cost upwards of $20,000. Moreover, even the full set of these many forensic tools does not cover the large set of cell phones available.

Naturally, many attempts have been made to facilitate the reverse engineering process. We discuss some of these attempts in Section 4. Because the process is error prone, even with commercial tools, many refer to the process of analyzing phone simple as *data recovery* rather than digital forensics. Here, we propose only data recovery techniques that are perhaps sufficient for the "fair probability" [1, 17] required of the *probable cause* standard used in obtaining search warrants. We do not assert our techniques are forensically valid or sufficient for cases requiring evidence that is beyond a reasonable doubt. In both cases, deeper validation for specific models would be required, which is beyond our scope.

2.1 Problem Definition

The goal of reverse engineering is to interpret data that has been constructed according to some unknown format specification S by observing representative examples of the data. Suppose we are given a set of *records* $R = \{r_1, r_2, \ldots, r_n\}$, each of which is an example of data laid out according to S. Each record r_i is composed of one or more non-overlapping *fields* such that $r_i = (f_1, f_2, \ldots, f_m)$. A field represents a meaningful set of data, such as an integer or a string. We assume that the boundaries between fields are not known and that there are no explicit field delimiters within the record. In other words, without knowing the format, each record appears to be a sequence of non-delimited binary data. For each record r_i, our goal is to provide a hypothesized interpretation r'_i. This interpretation includes the starting position s, length ℓ, and type t of each field in the record such that $r'_i = (f'_1, f'_2, \ldots, f'_m)$ where f'_j is the tuple $(s_j,\ \ell_j,\ t_j)$.

2.2 Ares Design

Ares takes as input a sequence of records created using some unknown format and a probabilistic grammar with which to describe these records. Its task is then to come up with the most likely field layout for each record according to the current grammar and then figure out which bytes in the format maintain constant values across records. The Ares algorithm is divided into two parts: *record-level analysis* and *cluster-level analysis*.

Record-level Analysis. Ares uses the Cocke-Younger-Kasami (CYK) [14] algorithm to determine the most likely interpretation of each record according to a probabilistic grammar. A grammar is composed of a set of symbols (terminals and non-terminals) and a set of rules that describe ways in which these symbols can be substituted for one another. In addition, each rule has an associated probability. Rules must be of the form $A \rightarrow BC$, which means that the symbol A can be substituted for the sequence of symbols BC. These rules must always have one symbol on the left-hand side and can have one or two symbols on the right. Each rule is given an estimated probability of occurring. The following is an example of the rules that would be used to represent a `UnicodeString` when parsing binary data. Symbols that represent our input data are enclosed in single quotes.

- `0.8 UnicodeString` → `UnicodeChar UnicodeString`
- `0.2 UnicodeString` → `UnicodeChar`
- `1.0 UnicodeChar` → `Ascii Null`
- `1.0 Null` → `'0x00'`
- `0.0093 Ascii` → `'0x20'`

- `0.0093 ...` *`[other Ascii entries]`*
- `0.0093 Ascii` → `‘0x7e’`

CYK is a bottom-up parsing algorithm that starts with the input data and repeatedly makes symbol substitutions until it reaches a special root symbol that represents the entire record. In this manner, a tree is constructed for each record that has the root symbol as its root and the original data as its leaves. The tree represents the most likely interpretation of the record. The CYK algorithm uses the probabilities associated with each rule to bias the resulting parse towards the most likely way of interpreting the input data. The probabilities assist CYK in determining which among several rules that match is the most likely parse, and therefore the precise values assigned are not critical, only their relative values. Experience and observation can drive the values chosen, but if a rule doesn't match, its assigned probability is not considered.

A fundamental limitation of using a grammar is that we cannot encode rules that are necessary for machine formats without a super-linear expansion of rules. For example, one field's *value* might express the *length* of another field. Our first extension to the CYK algorithm allows such conditions to be added to the production rules of a grammar to express relationships between its symbols. For example, it is possible to specify that a symbol must be within a certain range of numerical values or that the value of one non-terminal must be equal to the length of another. In this manner, Ares can recognize relationships between fields by only allowing conditioned substitutions to occur when their conditions are satisfied.

Our second extension to the CYK algorithm enables Ares to account for a change in endianness. While reading in the input grammar, every time Ares reads in a rule of the form $A \rightarrow BC$, it will also add the rule $A \rightarrow CB$ to its working set of rules with 30% lower probability. This allows Ares to take the little endian parse into consideration when applying the CYK algorithm.

The parse tree for a record actually has many levels of representation for a record. At the top level, it represents the record as a single symbol and at the bottom level it represents the record as each individual byte. In the middle of the tree is the representation we want: the record interpreted as a series of fields. The record-level analysis extracts this interpretation from the parse tree and outputs it as a sequence of tuples with the format `[(start, length), label]` where start is the byte offset of the field from the beginning of the record, length is the number of bytes in the field, and label is the inferred field type. Types we have defined include Unicode strings, ASCII strings, dates and times, phone numbers, and lengths of these fields. (Nokia has a proprietary format for dates and phone numbers that we do not detail here). If Ares is unable to match a sequence of bytes with a known type, it groups those into a single field and labels that field as `Binary`. Furthermore, the CYK algorithm ensures that each byte is assigned to exactly one field and that fields do not overlap.

Cluster-level Analysis. Ares' cluster-level analysis provides additional information about `Binary` fields, for which the boundaries and field types are unknown. In this stage of the algorithm, Ares records the range of values that bytes take on across multiple records. Any bytes that have a single value for every record can be considered *constants*.

To perform this analysis, Ares first groups record interpretations that have similar formats. Records are considered to have similar formats if they have the same sequence of field types. For example, if records r_i and r_j are both parsed as the sequence of fields `UnicodeString, Null, Binary` then Ares will place r_i and r_j in the same cluster. Note that the field lengths need not be the same across both records for them to be considered similar. In this way, the clustering is not perturbed by variable length fields. Once records with similar formats have been clustered, Ares compares the binary fields of records within each cluster and records the range of values for each byte within the field.

2.3 Challenges

Reverse engineering binary data using a sample-based approach presents a number of challenges. Several situations complicate the process by making it difficult to determine the boundaries in a record or the types of its fields.

Non-delimited Data. Boundaries between fields in binary data are rarely marked by delimiters in the data itself since any programs that are able to read the data in a meaningful way will know how many bits to read at a time. This lack of explicit delimiters makes it difficult to determine field boundaries when only the data is available. In reverse engineering, inferring these boundaries accurately is typically a prerequisite to determining higher level constructs such as relationships between fields and field semantics. In contrast, Ares essentially looks for semantics first and uses this knowledge to find field boundaries. Unfortunately, this means that Ares is only able to find boundaries where semantics can be ascertained.

Data Ambiguity. Ares relies on being able to find patterns in the input data that match rules in its grammar, but many types of fields are difficult to differentiate from one another. For example, suppose we encounter the byte pattern "0x00 0x41" in our input. This field could be interpreted as a single integer that stores the value 65 or it could be interpreted as the Unicode letter "A".

Ares resolves such ambiguous situations by calculating which case is more *probable* based on the probabilities given in its grammar. Ideally, we would obtain these probabilities by examining huge collections of binary data for which fields are known. Such collections do not exist to our knowledge, so we use imperfect estimates for the probabilities in our grammar. Our experience suggests that CYK is quite robust to the choice of probabilities given the good performance of Ares, presented in Section 5.

Data ambiguity can cause a number of difficulties. For example, it can cause Ares to make incorrect judgments about field boundaries. A single boundary error may then propagate across multiple fields.

Permutations. It is possible for the field ordering to change between records. Suppose we have two records $r_1 = (f_1, \ldots, f_i, f_j, \ldots, f_m)$ and $r_2 = (f_1, \ldots, f_j, f_i, \ldots, f_m)$. While the CYK algorithm will parse permutations correctly, Ares' cluster-level analysis algorithm will have a difficult time identifying that, for example, f_i has a constant value across records.

Limited Samples. The effectiveness of any sample-based approach is limited by the number and quality of available samples. Suppose there are two different possible field layouts

Phone	Access	Date Issued
Nokia 3360	TDMA	Q2 2001
Nokia 3200	GSM	Q4 2003
Nokia 6820	GSM	Q1 2004
Nokia 6230	GSM	Q2 2004
Nokia 6170	GSM	Q3 2004
Nokia 6101	GSM	Q3 2005

Figure 1: Information about the phone models used in our experiment

for a given type of input, yet our set of samples only represents one of these layouts. Ares has no way of knowing that the second layout is a plausible interpretation for this type of input.

Another problem arises from the fact that Ares only operates on positive examples. In other words, we do not give Ares any input that represents how a certain type of data should *not* be laid out. This fact hinders the cluster-level analysis algorithm that attempts to identify constants. We cannot conclude that this field will be constant for records we have not yet seen unless we have a sufficient set of negative examples. It has been shown that it is impossible to learn a language without both positive and negative examples [12].

3. EVALUATION

In this section, we evaluate Ares' accuracy when applied to cell phone call logs. We describe the construction of the grammar used in this experiment, provide a brief overview of our data collection process, and present quantitative results.

3.1 Methodology

To evaluate Ares, we created a grammar by manually reverse engineering the call logs on a Nokia 3360 phone, and we then tested the rule set on five other Nokia phone models — 3200, 6101, 6170, 6230, and 6820 — using two of each model. We did not modify the rules before examining the ten test phones. Figure 1 provides some details about the five phone models, which were released over a 5-year period. Note that Nokia uses a proprietary OS and chip set; existing binary instrumentation tools would be ineffective.

We selected the five test models from a large collection of used cell phones that we have purchased from resellers. Conveniently, many of these cell phones arrived with user data intact, including call logs. The models used in our experiments were selected for two reasons. First, we could obtain a raw memory dump using the Tornado UFS, a commercial phone flashing tool (used to unlock a phone from a carrier) [16]. Second, we could also obtain results from *Pandora's Box*, a commercial phone forensics tool, which we used as a point of comparison.

Unfortunately, for this preliminary work, we did not have the opportunity to test our rules on phones from other manufacturers. We were not able to verify that the rule set we constructed from the Nokia 3360 is transferable to another manufacturer's data set. We intend to perform this evaluation in future work.

For each phone model, Ares processed the extracted call logs as a series of *records*, each record representing a single entry in the call log. Fortunately, memory is acquired by Tornado UFS such that each call entry is already a distinct record. The first five entries of the call log from our Nokia 3360 phone appear in Figure 2. In all, Ares examined 482 call logs and 454 contact entries.

3.2 Evaluation

The grammar that we produced from manually reverse engineering the Nokia 3360 call log format consisted of roughly 800 rules representing several field types. (In fact, most of the rules are for terminals, such as one rule for each ASCII character.) The grammar included rules for Unicode strings, phone numbers, and dates and times. Our grammar also represents fields that denote the length of other fields. Section 2 shows several examples from the rule set.

Ares' task in this experiment was to produce the most likely interpretation of each call log entry for the five test phones given our manually constructed grammar. For each log entry, the output of Ares is a sequence of tuples that labels a byte range as a specific field type. The results are shown in Figure 3.

To evaluate accuracy, we compared Ares' output for each record with an assumed "correct" interpretation that we constructed using Pandora's Box. The programmers of Pandora's Box confirmed with us that they manually reverse engineered the format of each of the tested phones. As a result, this correct interpretation represents a lower-limit on the performance of manual reverse engineering and allows us to compare the accuracy of Ares to that of manual efforts. We used two different metrics when making this comparison: byte-based and tool-based.

Byte-based accuracy is the number of bytes in a record that Ares labeled correctly divided by the total number of bytes in the record. A byte is labeled correctly if Ares grouped it into the correct field type. For example, suppose the tuple `[(2, 8), UnicodeString]` appears in Ares' output for a single record. If the correct format is `[(2, 6), UnicodeString]` `[(8,2) Ascii]` then Ares labeled bytes 2 through 7 correctly, but bytes 8 and 9 are labeled incorrectly. Any bytes that Ares was unable to label are considered to be incorrectly labeled.

For records with fixed-length fields, this is a fairly good metric. However, the presence of variable-length fields can skew the byte-based accuracy significantly. For example, suppose our input records contained 4 fixed-length fields that Ares was unable to parse and 1 variable-length field V that Ares was able to parse successfully. Records in which V is 200-bytes long are going to have a significantly higher byte-based accuracy than those in which V is 5 bytes long.

Tool-based accuracy is the number of fields in a record that were correctly labeled divided by the number of fields recognized by Pandora's Box. We considered fields correctly labeled if they matched the Pandora's Box output. Any fields that Pandora's Box was unable to parse were ignored when calculating tool accuracy.

This metric serves as a reasonable comparison to manual reverse engineering accuracy. However, it also has limitations. Since the output of Pandora's Box consists only of the fields that its authors were able to manually reverse engineer, it may in fact lack some valuable information.

Discussion. We performed two sets of tests. We first evaluated Ares without our extension that accounts for endianess. The results are shown in Figure 3. Ares' byte-based accuracy is low for all models tested. Ares is only able to parse fields that represent name, name length, phone number, phone

```
0=05000C0705004F0065006400670061007200070B0100030755529540000813O207D5010F0F1F11010008130307D5010E12002801
1=07000E0708004D00640061006E00690065006C00080B0100030A4A15556427000813O207D501100F383001
2=0300090B01000A0B18A855545990001207020050006C007500630072006500630069006100081303O7D5010F0D151A01
```

Figure 2: The first three entries of a Nokia 3360 call log (phone numbers have been altered to preserve privacy of the original owner).

	Calls		Contacts	
Phone	**Byte**	**Tool**	**Byte**	**Tool**
Nokia 3360	42%	97%	67%	99%
Nokia 3200	53%	93%	51%	98%
Nokia 6101	57%	96%	63%	97%
Nokia 6170	29%	26%	64%	52%
Nokia 6230	39%	56%	33%	34%
Nokia 6820	54%	95%	69%	96%

Figure 3: Ares' accuracy using a grammar generated from the call log of a Nokia 3360 when applied to the call logs of five other models.

	Calls		Contacts	
Phone	**Byte**	**Tool**	**Byte**	**Tool**
Nokia 3360	42%	97%	67%	99%
Nokia 3200	54%	93%	51%	98%
Nokia 6101	58%	96%	63%	97%
Nokia 6170	**53%**	**65%**	64%	52%
Nokia 6230	**52%**	**81%**	**38%**	**62%**
Nokia 6820	54%	95%	69%	96%

Figure 4: Ares' accuracy using a grammar generated from the call log of a Nokia 3360 with automatically added rules for handling little endian. Differences with Fig. 3 are in bold. Note that no results for big endian phones are changed.

number length, and timestamps. Although these are important fields, a significant portion of the call log data is devoted to flags and other small fields that Ares is unable to parse and thus labels as `Binary` fields. Since we never count `Binary` fields as correct towards byte-based accuracy, our results are fairly low.

The tool-based accuracy is quite high for several of the models, meaning Ares was able to parse the call log data almost as effectively as Pandora's Box. Note that two of the models (6170 and 6230) have extremely low tool-based accuracy. This is due to an endianness difference. The 3360 from which the grammar was created stores its data in big endian format whereas these two models store their data in little endian. Ares relies on patterns that are described in its grammar and switching the endianness of the data invalidates many of these patterns. This difference caused a drop in overall accuracy for these two models.

In a second experiment, we evaluated the Ares' endianness extension for CYK. The results of this modification are shown in Figure 4. Our modification improved accuracy for the little-endian models (6170 and 6230) while maintaining the accuracy of the big endian models. This result demonstrates the extensibility of our approach. Note that the contact list parsing for the 6170 model phones is not improved; this is due to the stringency of our evaluation metrics. While Ares only misses the first letter of a Unicode string in the contact list, we consider the entire field incorrect.

Ares parses each record in 0.6 sec on average; each record is 60 bytes on average. The system's performance is restricted by CYK's $O(n^3)$ runtime for input of size n, but we expect more efficient parsing algorithms can be applied in future work. Moreover, the system can work on records in parallel. Ares is a minuscule system compared to the massive engineering efforts required for instrumentation and taint analysis platforms. The complete Ares system is written in only 1,800 lines of Python.

4. RELATED WORK

Related work on reverse engineering has focused on either protocol *message* formats, or memory and file *record* formats, but we treat them as equivalents. These tools are typically either *sample-based* or *instrumentation-based* — Ares is an example of a sample-based approach. We did not compare Ares against these works as each has a limitation or assumption that does not apply well to the phone domain.

Sample-based Approaches. Discoverer [8], a sample-based tool, attempts to automatically derive the format of messages sent by an application-level network protocol. Given a sample of application-level messages, Discoverer splits each message into tokens, clusters each token, and attempts to infer the token format by comparing it to other messages in the same cluster. While Discoverer is strictly limited to identifying fields as just one of two types — text or binary — Ares can handle innumerable and customizable field types using its extensible grammar.

LearnPADS [11] is a sample-based system used to automatically infer the format of ad hoc data, creating a specification of that format in the PADS data description language. LearnPADS begins by splitting the raw input data into a series of chunks, typically line-by-line or file-by-file. The chunks are then divided into tokens using a lexer. LearnPADS uses a histogram of the token frequencies to infer the structure of the data. Unfortunately, LearnPADS relies on explicit delimiters that are not commonly found in a phone's binary data. In contrast, Ares does not rely on delimiters for parsing.

Cozzie et al. [7] detect botnets using Bayesian unsupervised learning to locate data structures in memory. Unlike Ares, their approach is not designed to parse the data, and cannot manage length fields.

Instrumentation-based Approaches. Many approaches, including Polyglot [5], Tupni [9], and Dispatcher [4] require instrumentation of the binary executable — a complex process.

Polyglot [5] relies on executable analysis to reverse engineer application-level protocol formats. Polyglot was intended to overcome the deficiencies of sample-based approaches by observing how a program processed received messages. Tupni [9] uses taint analysis to reverse engineer input formats with high accuracy. The tool attempts to derive information such as field boundaries, record sequences, and field constraints. Dispatcher [4] also instruments executables to infer the format of messages sent by a program as well as the field semantics of both sent and received messages.

On a platform as volatile as cell phones where the hardware and software are constantly changing with each new model, instrumentation is a poorly-suited approach. Instrumentation is a challenging and time-consuming process that would have to be repeated for each different combination of architecture and OS. As we stated earlier, possession of the hardware, or an emulator of the hardware, is insufficient. Moreover, small changes in the OS (or architecture) requires modification of the instrumentation platform, and these systems are typically closed-source, proprietary, and obfuscated to prevent disclosure of commercial advantages. Note that manufacturers have a history of not cooperating, even with law enforcement, when it comes to unlocking, acquiring data, and parsing data from phones.

In sum, Ares' main advantage over other approaches is its ability to work adeptly and quickly with new and unseen phone models. We expect that a single, well-constructed grammar could be used across a welter of different phone models to provide swift analysis.

5. CONCLUSION

We have argued that reverse engineering of mobile phone data requires a data-driven approach. In contrast to recent works on malware analysis largely based on instrumentation, Ares uses a modified CYK parsing algorithm and an extended grammar to find field boundaries and infer simple relationships between fields. The grammar must be manually constructed, but can then be reused to parse novel data. The data-driven approach used in Ares is robust to CPU and OS changes and does not rely on delimiters. While there are limitations to Ares' approach, its accuracy is often on par with manual reverse engineering efforts.

We believe Ares can be based on a library of grammar sets that are focused on specific types of input data. For example, we imagine a set that works best on Nokia models and one for Samsung. Another grammar can be targeted for parsing data sent through network protocols. Moreover, we suspect that Ares' full potential is to be realized by programming it to adjust its output based on feedback from the user. For example, a user might recognize that the boundary between two fields in Ares' output is off by a few bytes. The user would provide Ares with feedback by telling it to fix the boundary in the correct position. Ares would then rerun the CYK algorithm with this fixed boundary and produce a better interpretation of the input data. By repeating this cycle, Ares could work in tandem with the user to produce a detailed answer with greater speed and accuracy than either party could produce on their own.

We view our efforts as an important first step in a promising approach. In future work, we intend to explore Ares' effectiveness at parsing more complex data types and relationships. We also plan to evaluate Ares using phones from other manufacturers besides Nokia. In preliminary tests, Ares was able to parse data on Motorola phones with minimal modification to the Nokia-based grammar.

6. ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation awards CNS-0905349 and DUE-0830876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

7. REFERENCES

[1] Illinois v. Gates, 462 U.S. 213, 238 (1983).

[2] R. Ayers, W. Jansen, A. Delaitre, and L. Moenner. Cell Phone Forensics Tools: An Overview and Analysis Update. Interagency report 7387, NIST, Feb 2007.

[3] S. Bhansali et al. Framework for instruction-level tracing and analysis of program executions. In *Proc. ACM Virtual Execution Environments Conf.*, pages 154–163, 2006.

[4] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. In *Proc. ACM CCS*, Nov 2009.

[5] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *Proc. ACM CCS*, pages 317–329, 2007.

[6] B. Chess. Improving Computer Security Using Extended Static Checking. In *Proc. IEEE Security & Privacy*, pages 160–176, See `http://fortify.com`, 2002.

[7] A. Cozzie, F. Stratton, H. Xue, and S. T. King. Digging for data structures. In *Proc. ACM OSDI*, 2008.

[8] W. Cui, J. Kannan, and H. J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In *USENIX Security Symp*, pages 1–14, 2007.

[9] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz. Tupni: automatic reverse engineering of input formats. In *Proc. ACM CCS*, 2008.

[10] K. Fisher, D. Walker, and K. Q. Zhu. LearnPADS: automatic tool generation from ad hoc data. In *Proc. ACM SIGMOD*, pages 1299–1302, 2008.

[11] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From dirt to shovels: fully automatic tool generation from ad hoc data. In *Proc. ACM POPL*, 2008.

[12] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[13] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 129 –142, 18-22 2008.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2000.

[15] W. Jansen and R. Ayers. Guidelines on Cell Phone Forensics. Technical Report NIST Special Publication 800-101, National Institute of Standards and Technology, Gaithersburg, MD, May 2007.

[16] K. Jonkers. The forensic use of mobile phone flasher boxes. *Digital Investigation*, 6(3-4):168 – 178, 2010. Embedded Systems Forensics: Smart Phones, GPS Devices, and Gaming Consoles.

[17] N. Judish et al. Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations. US Dept. of Justice, 2009.

[18] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proc. ACM PLDI*, pages 190–200, 2005.