The Analysis of Dimensionality Reduction Techniques in Cryptographic Object Code Classification

Jason L. Wright[1] and Milos Manic[2]
[1]Idaho National Laboratory, Idaho Falls, Idaho, USA,
[2]University of Idaho at Idaho Falls, Idaho Falls, Idaho, USA,
jlwright@ieee.org, misko@ieee.org

Abstract—**This paper compares the application of three different dimension reduction techniques to the problem of classifying functions in object code form as being cryptographic in nature or not. A simple classifier is used to compare dimensionality reduction via sorted covariance, principal component analysis, and correlation-based feature subset selection. The analysis concentrates on the classification accuracy as the number of dimensions is increased. It is demonstrated that when discarding 90% of the measured dimensions, accuracy only suffers by 1% for this problem. By discarding dimensions, computational intelligence techniques can be applied with a drastic reduction in algorithmic complexity. The primary focus is on Intel IA32 instruction set, but analysis shows consistent results on the Sun SPARC instruction set.**

Index Terms—**correlation-based feature subset selection, cryptography, dimensionality reduction, principal component analysis (PCA), sorted covariance.**

I. Introduction

Object code is the result of compilation of source code (C, C++, etc.). Source code is generally processor architecture independent, but the resulting object code is tied to a particular CPU instruction set architecture. Object code can also refer to a binary executable for a given operating system and instruction set architecture.

The location cryptography in compiled object code has three primary areas of application: malware analysis, hidden software feature detection, and export compliance. In the malware analysis community, location of cryptography is a concern as malware authors are employing strong cryptography primitives. The much publicized Conficker worm, for example, uses RSA and RC4 [1–2], and botnets and other forms of malware are using similar methods [3–4]. Hidden software feature detection is the detection of unpublished, possibly malicious, features in software. The idea here is to detect the presence of cryptography algorithms in unexpected places. For example, the Microsoft Calculator that ships with Windows should not contain cryptography; encryption is not one of its listed features. In the United States, an export license is required in many cases if cryptography is to be exported in source or binary form as a part of a product. To verify that cryptography is not being unintentionally exported, at least one company provides a service that will scan source code for known algorithms based on a database [5].

This work focuses on the problem of identifying the dimensions to be measured for accurate location of cryptography algorithms within a sample of compiled object code. Once a dataset is gathered, a the behavior of a simple classifier (linear regression) is used to compare and contrast three different dimensionality reduction techniques: sorted covariance, Principal Component Analysis, and Correlation-based Feature Subset selection (CFS).

A simplistic technique for identifying cryptography has been implemented for the IDAPro disassembler [6–7]. The constants used by various cryptographic algorithms are used as search strings. If identified, the assumption is made that the algorithm is present. References to the located string can then be used to determine the location of the cryptographic function within the binary. The problem with this technique is the assumption that the constants have not been tampered with.

In [8], a simplistic, expert driven approach was used to identify the properties of cryptographic functions. Empirically defined weights were applied to a single neuron with a linear activation function. The inputs were the density of a few opcodes (XOR, SHL, SHR, ROR, ROL). This method was refined to include the total number of each opcode as well as the density in training a non-trivial neural network [9].

In this work, analysis is done on the opcodes emitted by compilers to determine which instructions correspond most closely with cryptography. Initial work is performed with the Intel IA32 (x86) architecture [10–11]. IA32 is also classified as a Complex Instruction Set Computer (CISC). The same method is then used on the SPARC version 9 architecture [12]. SPARC is chosen because it is a representative Reduced Instruction Set Computer (RISC). Results should generalize to other instruction sets (PowerPC, ARM, etc.).

Various techniques have been applied to the problem of identifying malware. Several of the techniques have focused on statistical properties of malware. N-grams of opcodes [13] and Bayesian analysis [14] have been demonstrated as being capable of identify a sample of object code as being malicious. This work differs in that ultimate goal is classification of individual functions instead of the program as a whole.

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U. S. Department of Energy. The United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

II. Statistical Dimensionality Reduction

Informally, covariance is the measure of dependence of two variables [15]. A high positive covariance indicates that large and small values of one variable occur with large and small values of the other. If small values of one variable occur with large values of the other variable, the covariance will tend to be negative. A covariance close to zero means that the two variables do not possess a strong relationship. Formally, the covariance of two random values, X and Y is defined in (1) where μ_X and μ_Y are the mean of X and Y, respectively, and $p(x, y)$ is the joint probability mass function of x and y or the probability that $X = x$ and $Y = y$, (2).

$$\begin{aligned} \operatorname{cov}(X,Y) &= \sum_x \sum_y (x-\mu_X)(y-\mu_Y)p(x,y) && (1) \\ p(x,y) &= P(X = x \,\text{and}\, Y = y) && (2) \end{aligned}$$

A matrix of covariance values can be computed for n random variables X_i, X_j for $i = 1, \ldots, n$, $j = 1, \ldots, n$ as shown in (3). The resulting matrix is symmetric because $\operatorname{cov}(X, Y) = \operatorname{cov}(Y, X)$.

$$c_{i,j} = \begin{cases} \operatorname{var}(X) & i = j \\ \operatorname{cov}(X_i, X_j) & i \neq j \end{cases} \quad (3)$$

Taking a vector (row or column) from the matrix, the covariance with a particular variable can be determined. Sorting by the absolute value gives a ranking of the covariance of one variable to the variable of interest. This is a naïve form of dimensionality reduction by which the highest magnitude covariance dimensions are ranked against the classification dimension. It does not, however, take into account redunancy amongst the variables (those that have a covariance with each other).

Principal Components Analysis (PCA) is robust form of dimensionality reduction. The results of PCA are a linear transformation of the data to a new coordinate system. Given n dimensions, PCA produces n linear transforms of the original dimensions of the data as shown in (4), where pc_1 is the result of the linear transformation $a_{n,m}$ (n is the principal component number and m is the dimension number) and x_n is the value for dimension n for a given data point.

$$\begin{aligned} pc_1 &= a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n \\ &\vdots && (4) \\ pc_n &= a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n \end{aligned}$$

Each linear transformation is orthogonal. The first principal component captures the majority of the variance of the original data, and each successive principal component captures less. The total variance is equal to the original data variance. Dimensionality reduction is achieved by keeping some number of the principal components that capture the majority of the variance in the data set, and discarding the rest. Practically this means that all of the original dimensions must be measured for each data point, but only the most important principal component results need to be stored.

The last technique applied in this paper is that of Correlation-based Feature Subset selection (CFS) [16]. This algorithm finds dimensions that are predictive of the type of the object, but have little correlation with each other. Simple covariance analysis across the classification dimension does not capture the effect of covariance of other dimensions with each other. CFS seeks to address this problem by applying a greedy search of the covariance matrix to maximize covariance to the classification dimension and minimize the covariance of the selected variables with each other.

III. Dimensionality Reduction Analysis

The three dimension reduction techniques described in Section II (sorted covariance, PCA, and CFS) are applied to two different data sets. Performance is measured in terms of accuracy (AC), precision (P), False Positive rate (FP), and False Negative rate (FN). Each of these is defined in terms of a confusion matrix [17], like that in Table 1. Accuracy, (5), is a measure of correctly classified items (positive and negative). Precision, (6), is a measure of correctly classified positive cases. False positive, (7), and false negative, (8), measure the proportion of incorrectly classified positive and negative cases, respectively

classified as:		
non-crypto	**crypto**	
(a)	*(b)*	**non-crypto**
(c)	*(d)*	**crypto**

TABLE 1: Example confusion matrix.

$$\begin{aligned} AC &= \frac{a+d}{a+b+c+d} && (5) \\ P &= \frac{d}{b+d} && (6) \\ FP &= \frac{b}{a+b} && (7) \\ FN &= \frac{c}{c+d} && (8) \end{aligned}$$

The first step in the process is to gather a representative data set. It must be evaluated by an expert to determine which functions are cryptographic in nature. After some preliminary data analysis, each dimensionality reduction technique is applied and evaluated for its impact on accuracy, precision, false positive rate, and false negative rate.

A. Gathering Data Set

Various disassembly and reverse engineering tools are available. In this work, the GNU *objdump* utility is used to disassemble libraries and applications. The general method is to take a representative library and for each function to compute the total number and density of each opcode observed. Density is defined as the total number of a particular opcode divided by the total number of opcodes in the function, (9).

$$\rho_{i,f} = \frac{\#\text{ of opcode } i \text{ in function } f}{\text{total opcodes in function } f} \quad (9)$$

The library is augmented with expert knowledge of which functions are cryptography and which functions are not (a simple binary "yes" or "no" for each function in the library). The opcode totals and densities combined with the binary "iscrypto?" value forms the matrix on which this paper is based.

The representative library chosen is the C library from OpenBSD on the Intel (32 bit) platform (IA32). This library contains 1727 functions. Included in those functions are several cryptographic algorithms: SKIPJACK, BLOWFISH, DES, 3DES, MD5, MD4, SHA1, SHA2, etc. (23 of the 1727 functions). To increase the number of cryptographic algorithms, implementations of GOST, TEA, LUCIFER, RC5, RC4, and IDEA were taken from [18], compiled, and added to the base C library. System calls like *open()*, *close()*, *read()*, and *write()* were removed from the library. In total, 1776 functions were evaluated: 1723 non-cryptographic and 53 cryptographic.

The representative library was compiled with the GNU Compiler Collection (*GCC*) using four different compiler optimization levels (O0, O1, O2, and O3). Each level changes the scheduling of instructions, as well as ordering and may involve loop unrolling and function inlining. All of these may have an impact on the instruction counts and densities.

An oddity of the IA32 instruction set is that the cheapest way to zeroize a register is using the `xor` instruction, e.g. `xor %eax,%eax`. This is not truly an XOR operation; it is an assignment of zero to the register `%eax`. As a result, instructions using this idiom were not included in the data set except as a part of the total number of instructions in each function.

The definition of what, precisely defines an opcode is also somewhat ambiguous. Originally this work used strictly the mnemonic output by the disassembler to uniquely determine the opcode. For the IA32 instruction set, there are two commonly used assembly syntaxes, referred to as AT&T and Intel. The *objdump* disassembler uses AT&T syntax which adds a type modifier to the end of many instructions [19]. For example, `movb`, `movw`, `movl` are 8bit (byte), 16bit (word), and 32bit (long) movement instructions. In Intel syntax the mnemonic is simply `mov` for all three, because the width of the data type is inferred by the registers involved. The authors decided to translate AT&T syntax to Intel syntax.

The final list of instructions used and the opcode translations are shown in Table 2. For each function in the representative library, the total number of each opcode and the density (number of that particular opcode divided by the total number opcodes in the function) was computed.

B. Initial Classification

At this point, there are 94 measured dimensions for each function in the representative library: 47 totals and 47 densities. The addition of the classification dimension makes for 95 total dimensions. A simple linear regression, chosen for computational inexpensiveness, was performed on this data matrix.

Table 3 shows the resulting confusion matrix after several minor expert classification errors were fixed. The false negative rate is 53.9% as classified by the linear regression, but the overall accuracy is 98.2%. The poor performance on cryptography functions suggests that if a solution exists for robust classification of cryptography versus non-cryptography it is non-linear in nature. It is, at least, close to being a linear.

addl, addw, adc ⇒ add					
cmpb, cmpw, cmpl ⇒ cmp					
incb, incw, incl ⇒ inc					
movb, movw, movl ⇒ mov					
movsbl, movswl, cltd ⇒ movs					
movzbl, movzbw, movzwl ⇒ movz					
shrd, shrl ⇒ shr			andl ⇒ and		
decl ⇒ dec		divl ⇒ div		idivl ⇒ idiv	
negl ⇒ neg		pushl ⇒ push		roll ⇒ rol	
shld ⇒ shl		subl ⇒ sub		testb ⇒ test	
call	cld	imul	ja	jae	jb
jbe	je	jg	jge	jl	jle
jmp	jne	jns	js	lea	leave
nop	not	or	pop	repnz	ret
ror	sar	sete	setg	xchg	xor

TABLE 2: Opcodes measured for each function and aliases.

classified as:		
non-crypto	**crypto**	
6824 *(a)*	12 *(b)*	**non-crypto**
112 *(c)*	96 *(d)*	**crypto**
Classification rate		
0.18%		**false positive**
53.9%		**false negative**
98.2%		**accuracy**
88.9%		**precision**

TABLE 3: Linear regression results (94 measured dimensions).

Figure 1 shows the residual case order plot of the data. Of the 7048 points, 235 are identified as outliers by the residuals (outside the 95% confidence level); 128 of these outliers are misclassified by the linear regression and 107 are correctly classified outliers. In no case was an outlier identified using the residual that was not also misclassified by the linear regression. This suggests that there are extreme values in the outliers.

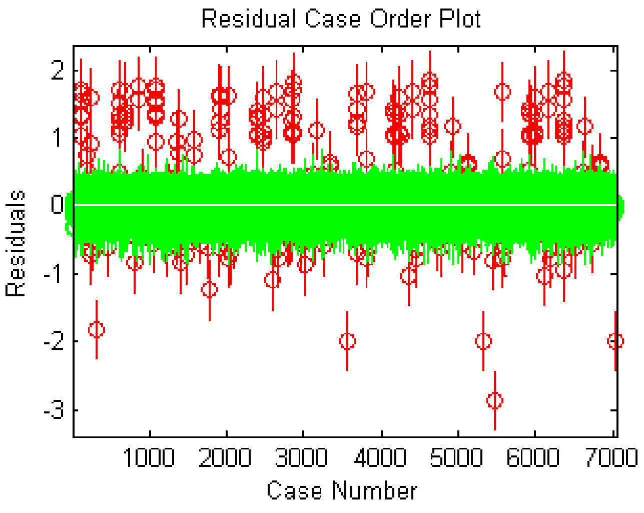


Fig. 1. Residual case order plot.

C. Dimensionality Reduction

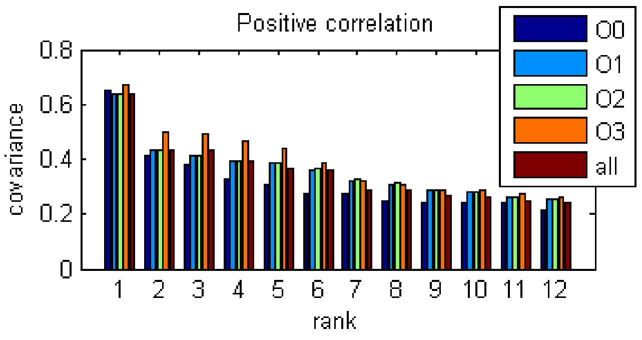
The data matrix was standardized to zero mean and unit standard deviation. Zero mean simplifies (1) by making

$\mu_X = \mu_Y = 0$. By standardizing, each dimension is given equal weight in the covariance (i.e. columns with high numerical values do not dominate the covariance).

Figure 2 depicts the twelve columns with the highest covariance of the data set to the expert determination. Figure 3 depicts the columns with the lowest covariance.

According to Figure 2, the density of `xor` instructions is the single most telling feature of the dataset followed closely by the total number of right shifts (`shr`). This makes intuitive sense as the `xor` operation is rarely used outside of cryptography and is quite common in the confusion part of of Feistel networks [20]. Other indicators are simple bitwise operations, like `and`, `or`, and `rol` (Rotate-Left) or arithmetic operations like `add`. One oddity is the `movz` (Move with Zero-extension) instruction. Upon closer inspection, this instruction is commonly used to compute modulo 256 or 65536. This is used in table references in cryptography algorithms like RC4 and others.

In Figure 3, no feature has a strong negative covariance with the expert determination. Several of these instructions make intuitive sense as well. For example the `call` instruction is used to call subroutines. Most core cryptography algorithms are written to avoid calling other subroutines.

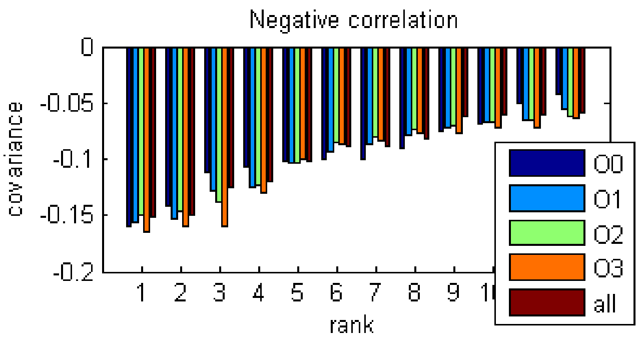


Rank	-O0	-O1	-O2	-O3	all
1	ρxor	ρxor	ρxor	ρxor	ρxor
2	σxor	σshr	σshr	σshr	σshr
3	σshr	σxor	σxor	σxor	σxor
4	σmovz	σmovz	σmovz	σmovz	σmovz
5	σor	σor	σor	$\sigma$and	σor
6	σadd	$\sigma$and	$\sigma$and	σor	$\sigma$and
7	$\sigma$and	ρor	ρor	σshl	ρor
8	ρlea	σshl	σshl	ρshr	σshl
9	σlea	ρrol	ρrol	ρrol	ρrol
10	σmov	σmov	ρshl	ρor	σrol
11	σrol	σrol	σrol	σadd	σmov
12	ρor	ρshl	σmov	σrol	ρshr

Fig. 2. Covariance analysis of positive features.

Using the standardized data matrix, a principal components analysis is performed. Figure 4 is a Pareto chart of the first ten principal components (PCs) and the amount of variance in the data set that is captured by those components. There is a bar for each of the ten PCs and the line is a running some of the variance captured. In a well behaved data set, the first few PCs would capture 80% or more of the total variance of the set. However, in this set, the first 10 PCs represent less than 45% of the variance of the data set.

With so little variance captured in each principal component, an examination of the weights assigned to each input dimension is difficult to interpret. Again, in a well



Rank	-O0	-O1	-O2	-O3	all
1	ρpush	ρcall	ρtest	ρcall	ρcall
2	ρcall	ρpush	ρcall	ρtest	ρpush
3	ρsub	ρje	ρpush	ρpush	ρtest
4	ρje	ρtest	ρje	ρje	ρje
5	ρleave	ρleave	ρleave	ρleave	ρleave
6	ρjne	ρret	ρret	ρjne	ρjne
7	ρcmp	ρjne	ρjne	ρret	ρret
8	ρret	ρsub	ρjmp	ρjmp	ρsub
9	ρtest	σtest	σtest	σcall	σtest
10	σjne	σcall	ρsub	ρsub	σje
11	σje	σje	σcall	σtest	ρjmp
12	σcmp	ρjmp	σje	σje	σjne

Fig. 3. Covariance analysis of negative features.

behaved data set, examination of the weights assigned to each dimension in each PC would provide insight into the behavior of each dimension and provide an indication of its relative importance.

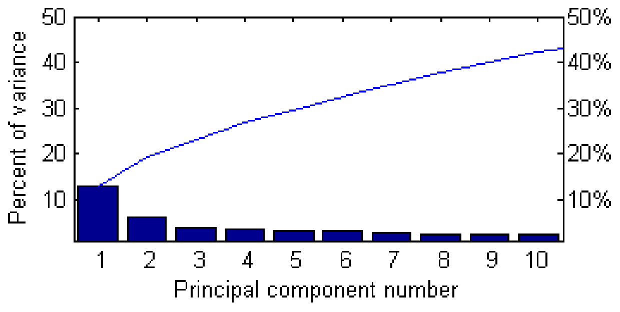


Fig. 4. Variance captured per principal component.

Finally, Correlation-based feature subset is performed. This technique attempts to find features that are predictive to the type of the object, but with little correlation with each other. The results of this analysis are shown in Table 4. CFS is compared with the covariance analysis already discussed for several optimization levels (O0, O2, and the whole data set). To obtain a ranking of attributes, 10-fold cross validation was used with CFS and rank is determined by the number of folds for which the attribute was selected.

All of the attributes selected for the whole data set also appear in the top covariance ranking. An interesting quirk, however, is that CFS picked exactly one negative covariance dimension: *ρcall*. It was only selected for the O2 optimization level.

Figure 5 shows the linear regression classification results of selecting attributes based on sorted covariance, PCA, and CFS. In the graphs, accuracy, precision, false positive rate, and false negative are plotted versus the number of dimensions used (X axis). The dotted line (COV) repre-

-O0		-O2		All	
COV	**CFS**	**COV**	**CFS**	**COV**	**CFS**
ρxor	ρadd/10	ρxor	ρxor/10	ρxor	ρje/10
σxor	ρxor/10	σshr	ρrol/10	σshr	ρshr/10
σshr	σxor/10	σxor	σxor/10	σxor	ρxor/10
σmovz	ρjne/9	σmovz	ρjle/9	σmovz	ρmovz/10
σor	σrol/9	σor	σmovz/9	σor	σrol/10
σadd	ρshr/8	σand	ρshr/8	σand	σxor/10
σand	ρlea/6	ρor	ρtest/8	ρor	σshl/7
ρlea	σmovz/6	σshl	ρcall/7	σshl	ρor/4
σlea	ρje/5	ρrol	σshl/7	ρrol	—
σmov	ρor/5	ρshl	ρxchg/6	σrol	—
σrol	ρmovz/4	σrol	ρor/2	σmov	—
ρor	ρpop/4	σmov	σshr/2	ρshr	—

TABLE 4: Comparison of covariance analysis to CFS analysis.

sents attributes sorted in order by their covariance with the classification dimension. The solid line represents the attributes selected by CFS. The dashed line is the number of principal components used for the linear regression A logarithmic scale is chosen for the number of dimensions axis to emphasize the shape of the curve along the lower number of dimensions. There is no improvement in false positive rate for CFS with more than 7 dimensions, and little improvment (0.2%) in accuracy with more than the 8 dimensions identified by CFS. In the case of false negative rate, CFS and covariance are closely matched at eight dimensions (within 0.5%), but adding more dimensions with covariance drops the false negative rate by approximately 5%. PCA has a lower false positive rate until almost all PCs are used. However, it maintains a consistently higher false negative rate until approximately the same number of PCs are used. PCA also lags behind both COV and CFS on accuracy. Based on these results, it would be difficult to pick a "best" dimensionality reduction technique; each one has its strengths.

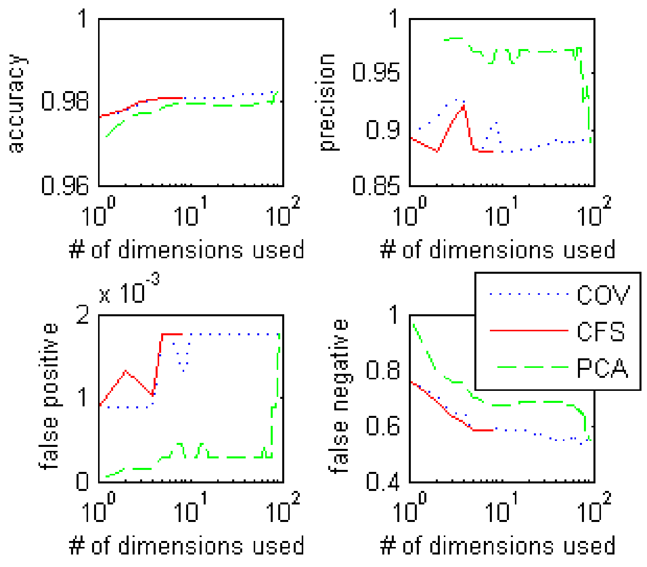


Fig. 5. Covariance, PCA, CFS effect on AC, P, FN, and FP (IA32).

IV. Test Results

To test the methodology of dimensionality reduction for consistency, the same techniques were applied to the SPARC version 9 instruction set. SPARC is a Reduced Instruction Set Computer whereas Intel IA32 is a Complex Instruction Set Computer. The same representative library used in Section III-A was compiled for SPARC using the same compiler as before: GCC. Four different optimization levels (O0 through O3) were applied as before.

The library was disassembled with *objdump*. Each instruction used in a cryptographic function was counted and the density was computed. As with IA32, *objdump* uses AT&T syntax and the type specifiers were removed from the instructions so that, for example, `ldb` (Load Byte), `ldh` (Load Halfword), `ldw` (Load Word), and `ldx` (Load Extended Word) are counted as `ld` (Load Integer). Several other translations were performed as well, including conditional move instructions like `movge` (Move, if Greater or Equal) being translated to simply `mov` and `addcc` (Add and set Condition Codes) being translated to `add`. The final list of instructions and translations is shown in Table 5. In all, 49 different opcodes were measured resulting in 98 dimensions (count and density for each).

movcc, movcs, move, movg, movge, movl, movle, movne ⇒ mov					
clrb, clrh, clrx ⇒ clr		andcc ⇒ and			
addc, addcc ⇒ add		ldsb, ldsw ⇒ lds			
ldub, lduh ⇒ ldu		stb, sth, stx ⇒ st			
ldx ⇒ ld		mulx ⇒ mul		orcc ⇒ or	
sllx ⇒ sll		srax ⇒ sra		srlx ⇒ srl	
subc ⇒ sub		udivx ⇒ udiv		andn	b
bcc	bcs	be	bg	bge	bgu
bl	ble	bleu	bne	bneg	bpos
brnz	brz	btst	call	cmp	inc
neg	nop	orn	restore	ret	retl
rett	save	sdiv	sethi	smul	wr
xnor		xor			

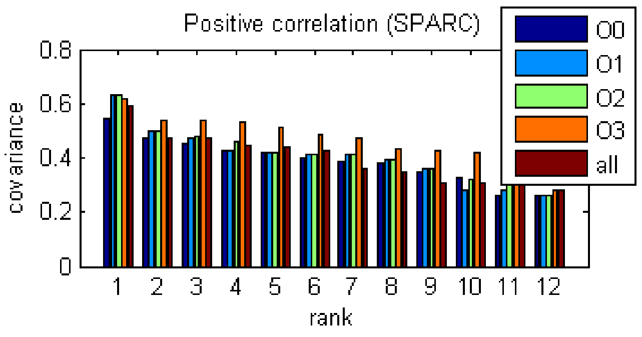
TABLE 5: Opcodes measured for each function and aliases (SPARC).

Figure 6 shows the covariance analysis of positive features with the classification label. The instructions that have high covariance with the classification dimension are quite different from IA32, but there are some similarities. The `ldu` (Load Unsigned) instruction figures highly in the covariance of all optimization levels. Like IA32, `xor` (eXclusive OR) and `sll` (Shift Left, Logical) also rank quite high.

classified as:		
non-crypto	**crypto**	
7613	0	**non-crypto**
75	141	**crypto**
Classification rate		
0%		**false positive**
34.72%		**false negative**
99.0%		**accuracy**
100%		**precision**

TABLE 6: Linear regression results for SPARC (98 dimensions).

Figure 7 shows the effect on a linear regression classifier based on dimension selection using sorted covariance, CFS and PCA. The accuracy of CFS closely follows sorted covariance and adding 89 more dimensions only improves it by 0.2%. Similiarly for false negative rate, sorted covariance and CFS follow a similiar curve with only slight improvement as additional dimensions are added. PCA



Rank	-O0	-O1	-O2	-O3	all
1	ρld	ρldu	ρldu	ρldu	ρldu
2	ρldu	ρsll	ρsll	σsll	σsll
3	σst	σsll	ρxor	ρxor	ρsll
4	σld	ρxor	σsll	ρsll	ρxor
5	σsll	$\sigma$and	$\sigma$and	$\sigma$and	$\sigma$and
6	$\sigma$and	σsrl	σsrl	σsrl	σsrl
7	σsrl	σldu	σldu	σxor	σxor
8	ρst	σxor	σxor	σor	σor
9	ρxor	σor	σor	σldu	ρld
10	ρsll	σld	σld	σld	σldu
11	σldu	ρor	ρor	ρor	σld
12	σor	σadd	σadd	σadd	ρor

Fig. 6. Covariance analysis of positive features on SPARC.

shows inverse behavior on the FP and FN graphs: FP grows as PCs are added (slowly) and falls on FN. Interestingly, precision for PCA is consistently higher than the other techniques.

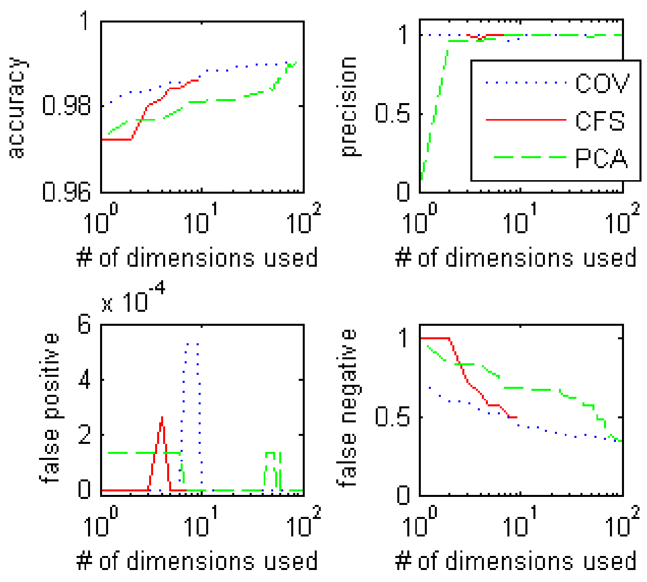


Fig. 7. Covariance, PCA, CFS effect on AC, P, FN, and FP (SPARC).

V. Conclusion

A method for automatically determining the correct opcodes to measure for the problem of identifying cryptographic functions in object code has been presented. The method was first applied to the Intel IA32 instruction set, a complex instruction set computer, and then applied to a reduced instruction set computer: SPARC. In both cases, accuracy was demonstrated to be less than one percent different after removing 90% of the measured dimensions. With the reduced dimensionality, computational intelligence algorithms can be applied without wasting time on dimensions which add no useful information. This can be a substantial computational savings as dimensionality is usually a multiplicative factor in the algorithmic complexity of computational intelligence techniques.

In previous work, the dimensions of the problem were expert defined, but the method presented here provides an automatic, statistically driven method for identifying those opcodes most indicative of cryptographic functionality. The only task left for the human is the expert determination of the cryptographic nature of each of the test samples.

References

[1] P. Porras, H. Saidi, and V. Yegneswaran, "An analysis of conficker's logic and rendezvous points," SRI International, Tech. Rep., March 2009. [Online]. Available: http://mtc.sri.com/Conficker/

[2] ——, "Conficker c p2p protocol and implementation," SRI International, Tech. Rep., September 2009. [Online]. Available: http://mtc.sri.com/Conficker/P2P/index.html

[3] J. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: overview and case study," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1.

[4] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 10–10.

[5] "Export KnowledgeBase," Black Duck Software. [Online]. Available: http://www.blackducksoftware.com/export/encryption-source-code

[6] I. Guilfanov, "FindCrypt," January 2006, http://hexblog.com/2006/01/findcrypt.html.

[7] ——, "FindCrypt2," February 2006, http://hexblog.com/2006/02/findcrypt2.html.

[8] J. L. Wright, "Finding cryptography in object code," in *Security Education Conference Toronto (SecTOR)*, October 2008.

[9] J. Wright and M. Manic, "Neural network approach to locating cryptography in object code," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, September 2009.

[10] *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference A–M*, Intel Corporation, September 2009. [Online]. Available: http://www.intel.com/products/processor/manuals/index.htm

[11] *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference N–Z*, Intel Corporation, September 2009. [Online]. Available: http://www.intel.com/products/processor/manuals/index.htm

[12] D. L. Weaver and T. Germond, Eds., *The SPARC Architecture Manual (Version 9)*. SPARC International, Inc., 2000.

[13] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," in *Proceedings of the 1st European Conference on Intelligence and Security Informatics EuroISI*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 204–215.

[14] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[15] J. L. Devore, *Probabiliy and Statistics for Engineering and the Sciences*, 3rd ed. Duxbury Press, 1991, ch. Expected Values, Covariance, and Correlation, pp. 200–206.

[16] M. A. Hall, "Correlation-based feature subset selection for machine learning," Ph.D. dissertation, University of Waikato, 1999.

[17] R. Kohavi and F. Provost, "Editorial: Glossary of terms," *Machine Learning: Special Issue on Applications of Machine Learning and the Knowledge Discovery Process*, vol. 30, no. 2-3, 1998.

[18] B. Schneier, *Applied Cryptography*. John Wiley and Sons, 1996.

[19] *GNU Assembler (as) Version 2.20*, Free Software Foundation. [Online]. Available: http://sourceware.org/binutils/docs-2.20/

[20] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed. CRC Press, August 2001.