# Students and Project Topics, 2023

Mark Stamp

**Master's Students**

- (CS 297, Spring '23) Anant Shukla (anant.shukla@sjsu.edu) — Bot detection
- (CS 297, Spring '23) Ritik Mehta (ritik.mehta@sjsu.edu) – TBD
- (CS 298, Spring '23) Lei Zhang (lei.zhang04@sjsu.edu) — TBD
- (CS 298*, Spring '23) Maithili Kulkarni (maithilivinayak.kulkarni@sjsu.edu) — Explainability
- (CS 298*, Spring '23) Atharva Sharma (atharva.sharma@sjsu.edu) — Concept drift
- (CS 298*, Spring '23) Inderpreet Singh (inderpreet.singh@sjsu.edu) — Adversarial attacks
- (CS 298*, Spring '23) Brooke Dalton (brooke.dalton@sjsu.edu) — Learning an algorithm

**Topics**:

The following are possible masters project topics, listed in no particular order. Some of these are fairly vague and some are more specific, but each could form the basis for a solid project.

1. Compute eigenvectors of call graphs (and possibly other graphs associated with executable code) to develop a score for malware. Here's the abstract from an unpublished manuscript that I recently reviewed.

   > Although the existing malware classifying and detecting algorithms based on graphs analysis can alleviate the negative influence of code obfuscation, they are confronted with two shortcomings. On one hand, the traditional graph matching techniques, such as GED (graph edit distances), graph

> isomorphism, and MCS (Maximum Common Subgraphs), often have too high time complexity; on the other hand, vertex-matching or edge-matching techniques, such as graph coloring or labeling, often have low correctness when two graphs contain a different type of function. To overcome these weaknesses, this paper introduces a novel malware classifying and detecting method based on static analysis of function-call graph constructed from assemble code. The proposed method convents the architectural feature of a graph into eigenvectors of graph by transition probability. In order to reduce the time complexity, the similarity of two graphs is computed by a method based on LCS (Longest Common Subsequence) instead of the existing graph matching techniques. On the premise of keeping the high accuracy of classifying and detecting malwares, the comparative experiment results show that the presented method has relatively low time complexity. This work reveal that the eigenvectors of function-call graph can represent the characteristics of a malware or a malware family exactly and uniquely.

As written, much of this abstract is nonsense, e.g., the last sentence. And, in fact, the paper does *not* actually use any eigenvector analysis. But, the basic idea they describe sounds like a good one, and would likely lead to a useful score that could be applied to difficult malware detection problems. A previous student project successfully applied eigenvector analysis to the malware detection problem `http://link.springer.com/article/10.1007/s11416-013-0193-4` This previous work applied directly to raw binaries. By focusing on certain types of graphs associated with the code, we should be able to obtain stronger scores.

2. Recently, one of my master's students developed a metamorphic worm and we published a nice paper with his results: Metamorphic worm that carries its own morphing engine, S. M. Sridhara and M. Stamp, *Journal of Computer Virology and Hacking Techniques* 9(2):49–58, 2013 `http://link.springer.com/article/10.1007/s11416-012-0174-z` We refer to the resulting worm as MWOR. Here's the abstract from that paper.

> Metamorphic malware changes its internal structure across generations, but its functionality remains unchanged. Well-designed metamorphic malware will evade signature detection. Recent research has revealed techniques based on hidden Markov models (HMMs) for detecting many types of metamorphic malware, as well as techniques for evading such detection. A worm is a type of malware that actively spreads across a network to other host systems. In this project we design and implement a prototype metamorphic worm that carries its own morphing engine. This is challenging, since the morphing engine itself must be morphed across replications, which imposes restrictions on the structure of the worm. Our design employs previously

developed techniques to evade detection. We provide test results to confirm that this worm effectively evades signature and HMM-based detection, and we consider possible detection strategies. This worm provides a concrete example that should prove useful for additional metamorphic detection research.

Some recently-developed techniques have had good success detecting the MWOR worms. We'd like to strengthen this worm so that it evades detection by these techniques. We have some ideas how to do so, and we are confident that it will succeed. Specifically, it seems that a significant degree of transposition will defeat the structural entropy technique. Also, techniques based on call graph analysis have proven successful at detecting MWOR worms. Heavy use of inlining and outlining should defeat call graph techniques.

3. A recent (and excellent) master's project used a score based on "structural entropy" for malware detection: `http://link.springer.com/article/10.1007/s11416-013-0185-4` Another related project used compression ratios as a basis for scoring. Since compression ratio depends on the entropy, these scores give similar results. In the latter paper, a lossless compression technique was used (Lempel-Ziv, which is often used to compress data files). It would be interesting to analyze similar techniques based on lossy compression algorithms (which are often used for multimedia files). Lossy compression might have the effect of minimizing some types of code obfuscation techniques, thereby yielding stronger results.

4. In some applications (e.g., metamorphic malware detection or software piracy detection) we need to compare two pieces of software and determine how "close" they are to each other. Previous masters students have studied many different software similarity measures. Here is an idea for a new similarity measure based on the LLL lattice reduction technique. Note that the LLL technique is used to attack the knapsack cryptosytem, and that attack is easy to code (it is covered in Chapter 6 of your *Information Security* book and also in the *Applied Cryptanalysis* book).

   Suppose we have $n$ malware samples, all from a given metamorphic family. We extract the `.text` section from each. Let $m$ be the length of the largest of these (in terms of the number of bytes), and pad the files with 0 so that they are all of length $m$. Denote these padded files as $M_1, M_2, M_3, \ldots, M_n$. Form the $m \times n$ matrix

$$T = [M_1\ M_2\ \cdots\ M_n]$$

   Note that the $M_i$ form the columns of $T$.

   Given a program $P$ that we want to score, extract its text section and pad (or truncate) so that it is of length $m$. We view $P$ as an $m \times 1$ matrix. If $P$ is a member of the metamorphic family, then we might expect

$$a_1 M_1 + a_2 M_2 + \cdots + a_n M_n \approx P \tag{1}$$

for some choice of $a_i \in \{0, 1\}$. Let

$$U = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

Then (1) can be written as

$$T \cdot U = P$$

where we have replaced the " $\approx$ " with " $=$ ". Mimicking the attack on the knapsack cryptosystem, we apply the LLL lattice reduction technique to the matrix

$$\mathcal{M}_{(n+m)\times(n+1)} = \begin{bmatrix} I_{n\times n} & 0_{n\times 1} \\ T_{m\times n} & -P_{m\times 1} \end{bmatrix}$$

If (1) holds true, then we expect the columns (within the first $n$ rows) of the resulting matrix to consist of "short" vectors in the lattice. So, if the program $P$ belongs to the metamorphic family, we expect to obtain short vectors in the lattice, but if $P$ is not a member of the family, this will not be the case. Hence, we may be able to use this technique to distinguish between benign files and members of the metamorphic family.

A couple of students have looked at this, without success. So, it might be a difficult problem.

5. Analysis of metamorphic detection techniques: First, develop a metamorphic generator that uses a variety of morphing (transposition, substitution, insertion, deletion) in controlled/variable amounts. Then test a variety of detection techniques. The goal is to determine the relative strengths and weaknesses of each detection strategy on specific morphing strategies (and combinations thereof). It would be best to implement your morphing engine within the LLVM complier framework. For more information, see Teja's paper from Spring 2013 (on generating metamorphic code using LLVM) and Michael's CS 297 work from Spring 2013 (a fancy morphing technique in LLVM).

6. Kullback-Liebler (KL) divergence for malware detection: KL divergence can be used to distinguish data that comes from different distributions. It has been used successfully in masquerade detection to separate "attack" commands from "padding" commands. The same general approach should work to improve malware detection,

where the goal is to identify and remove dead code that has been inserted to hide the statistical properties of the malware. For example, dead code insertion is an effective strategy to mask the statistical properties of metamorphic malware. Detecting dead code based on program flow and such is essentially a hopeless task. However, identifying dead code based on its statistical properties should be much more effective. See Geetha's paper from Spring 2013 for an application of KL divergence to the masquerade detection problem. Note that we could use a hill-climb with KL-divergence as the score to separate the data. Perhaps the "simple substitution" distance might also apply.

7. Develop and analyze encryption strategies for streaming media, with the emphasis on mobile platforms: Here, we are concerned with the tradeoff between security and efficiency. This work was started in a previous master's project a couple of semesters ago (Jaie Patil), but there is a lot more to do.

8. Analyze polymorphic malware using Gayathri's "simple substitution distance". This score should be particularly effective against polymorphic malware, provided the encryption is the equivalent of a simple substitution (e.g., XOR of a constant byte is the same as a simple substitution). The trick would be getting this score to work on byte values, as opposed to opcodes. That is, we can view the problem as one of scanning for an encrypted "signature" (or sorts). If the encryption is equivalent to a simple substitution, this distance measure should give us good results.

9. Recently, a technique was proposed to detect metamorphic engine properties directly from the generated metamorphic code. This is similar to some of the metamorphic detection strategies that previous students have worked on, but the focus here is slightly different. Here is the abstract from the paper "Detecting machine-morphed malware variants via engine attribution":

   > One method malware authors use to defeat detection of their programs is to use morphing engines to rapidly generate a large number of variants. Inspired by previous works in author attribution of natural language text, we investigate a problem of attributing a malware to a morphing engine. Specifically, we present the malware engine attribution problem and formally define its three variations: MVRP, DENSITY and GEN, that reflect the challenges malware analysts face nowadays. We design and implement heuristics to address these problems and show their effectiveness on a set of well-known malware morphing engines and a real-world malware collection reaching detection accuracies of 96% and higher. Our experiments confirm the applicability of the proposed approach in practice and indicate that engine attribution may offer a viable enhancement of current defenses against malware.

I'd like to see a comparison of the method(s) proposed in this paper to some of the techniques previously developed and analyzed by students (HMM, structural entropy, and so on), particularly for some advanced metamorphic generators—which have also been developed/analyzed by previous students.

10. A fairly straightforward malware detection technique based on call graph analysis is discussed in the paper, "A similarity metric method of obfuscated malware using function-call graph". The abstract for the paper is given below.

    > Code obfuscating technique plays a significant role to produce new obfuscated malicious programs, generally called malware variants, from previously encountered malwares. However, the traditional signature-based malware detecting method is hard to recognize the up-to-the-minute obfuscated malwares. This paper proposes a method to identify the malware variants based on the function-call graph. Firstly, the function-call graphs were created from the disassembled codes of program; then the caller–callee relationships of functions and the operational code (opcode) information about functions, combining the graph coloring techniques were used to measure the similarity metric between two function-call graphs; at last, the similarity metric was utilized to identify the malware variants from known malwares. The experimental results show that the proposed method is able to identify the obfuscated malicious softwares effectively.

    I would like to see tests of this technique versus advanced metamorphic malware. Then, assuming the technique is successful, we would like to study morphing strategies that would be effective at evading this type of detection. That is, we want to test the technique, then break it.

11. Call-graph analysis for malware detection is also considered in this recent paper "HDM-Analyser: a hybrid analysis approach based on data mining techniques for malware detection". Here is the (rather long and rambling) abstract:

    > Today's security threats like malware are more sophisticated and targeted than ever, and they are growing at an unprecedented rate. To deal with them, various approaches are introduced. One of them is Signature-based detection, which is an effective method and widely used to detect malware; however, there is a substantial problem in detecting new instances. In other words, it is solely useful for the second malware attack. Due to the rapid proliferation of malware and the desperate need for human effort to extract some kinds of signature, this approach is a tedious solution; thus, an intelligent malware detection system is required to deal with new malware threats. Most of intelligent detection systems utilise some data mining

> techniques in order to distinguish malware from sane programs. One of the pivotal phases of these systems is extracting features from malware samples and benign ones in order to make at least a learning model. This phase is called "Malware Analysis" which plays a significant role in these systems. Since API call sequence is an effective feature for realising unknown malware, this paper is focused on extracting this feature from executable files. There are two major kinds of approach to analyse an executable file. The first type of analysis is "Static Analysis" which analyses a program in source code level. The second one is "Dynamic Analysis" that extracts features by observing program's activities such as system requests during its execution time. Static analysis has to traverse the program's execution path in order to find called APIs. Because it does not have sufficient information about decision making points in the given executable file, it is not able to extract the real sequence of called APIs. Although dynamic analysis does not have this drawback, it suffers from execution overhead. Thus, the feature extraction phase takes noticeable time. In this paper, a novel hybrid approach, HDM-Analyser, is presented which takes advantages of dynamic and static analysis methods for rising speed while preserving the accuracy in a reasonable level. HDM-Analyser is able to predict the majority of decision making points by utilising the statistical information which is gathered by dynamic analysis; therefore, there is no execution overhead. The main contribution of this paper is taking accuracy advantage of the dynamic analysis and incorporating it into static analysis in order to augment the accuracy of static analysis. In fact, the execution overhead has been tolerated in learning phase; thus, it does not impose on feature extraction phase which is performed in scanning operation. The experimental results demonstrate that HDM-Analyser attains better overall accuracy and time complexity than static and dynamic analysis methods.

Your first step would be to understand what they are doing (which looks to be a not-too-complex data mining strategy). Then we would proceed as in Problem 11 above, that is, test the technique and then break it.

12. Malware and/or malware detection on mobile devices. There are many unique malware-related issues on mobile devices. For example, detection can be more challenging, since resources are so constrained—it is generally too costly to constantly scan for malware. From the malware writer's perspective, there are some interesting advantages, such as ways to spread the malware that are not so readily available in the non-mobile case. There are many, many possible project topics in this area. For example, you could try to develop metamorphic malware for a specific platform. Or you could try to adapt some previously-developed advanced detection techniques to

a (specific) mobile environment, with an emphasis on efficiency. As an example of published research in this area (of which there does not seem to be a whole lot), see the article “Symbian worm Yxes: towards mobile botnets?”; the abstract is below.

> In 2009, a new Symbian malware named SymbOS/Yxes was detected and quickly hit the headlines as one of the first malware for Symbian OS 9 and above all as the foretaste of a mobile botnet. Yet, detailed analysis of the malware were still missing. This paper addresses this issue and details how the malware silently connects to the Internet, installs new malware or spreads to other victims. Each of these points are illustrated with commented assembly code taken from the malware or re-generated Symbian API calls. Besides those implementation aspects, the paper also provides a global overview of Yxes’s behaviour. It explains how malicious remote servers participate in the configuration and propagation of the malware, including Yxes’s similarities with a botnet. It also tries to shed light on some incomplete or misleading statements in prior press articles. Those statements are corrected, based on the reverse engineering evidence previously. Finally, the paper concludes on Yxes’s importance and the lack of security on mobile phones. It also indicates several aspects future work should focus on such as communication decryption, tools to analyze embedded malware or cybercriminals motivations.

13. Malware that attacks during AV updates: Apparently, some AV services are disabled during updates. How best to take advantage of this (from the attacker’s point of view)? There is a recent (mediocre) article on this topic, “Antivirus security: naked during updates”; the abstract is below.

> The security of modern computer systems heavily depends on security tools, especially on antivirus software solutions. In the anti-malware research community, development of techniques for evading detection by antivirus software is an active research area. This has led to malware that can bypass or subvert antivirus software. The common strategies deployed include the use of obfuscated code and staged malware whose first instance (usually installer such as dropper and downloader) is not detected by the antivirus software. Increasingly, most of the modern malware are staged ones in order for them to be not detected by antivirus solutions at the early stage of intrusion. The installers then determine the method for further intrusion including antivirus bypassing techniques. Some malware target boot and/or shutdown time when antivirus software may be inactive so that they can perform their malicious activities. However, there can be another time frame where antivirus solutions may be inactive, namely, during the

time of update. All antivirus software share a unique characteristic that they must be updated at a very high frequency to provide up-to-date protection of their system. In this paper, we suggest a novel attack vector that targets antivirus updates and show practical examples of how a system and antivirus software itself can be compromised during the update of antivirus software. Local privilege escalation using this vulnerability is also described. We have investigated this design vulnerability with several of the major antivirus software products such as Avira, AVG, McAfee, Microsoft, and Symantec and found that they are vulnerable to this new attack vector. The paper also discusses possible solutions that can be used to mitigate the attack in the existing versions of the antivirus software as well as in the future ones.

14. Develop a practical system to detect cheating during online tests. This would be a continuation of Sumit's work from a couple of semesters ago. You can find his master's report at `http://scholarworks.sjsu.edu/etd_projects/243/`.

15. Comparison of static (and, possibly, dynamic) "software birthmark" techniques: These are somewhat like an implied watermark and can be used, for example, to identify pirated software. Implement various detection techniques and test their effectiveness by measuring the "resilience" and "credibility" (false positive/false negative rates) when metamorphic techniques are applied to the base software. A recent student (Shabana) did nice work on this topic; her project report is here: `http://scholarworks.sjsu.edu/etd_projects/236/` We've also written a soon-to-be published paper based on this project, which is lot shorter and easier to read—email me if you want to see it.

16. Here is the abstract from the paper, SAS: Semantics Aware Signature Generation for Polymorphic Worm Detection.

    String extraction and matching techniques have been widely used in generating signatures for worm detection, but how to generate effective worm signatures in an adversarial environment still remains challenging. For example, attackers can freely manipulate byte distributions within the attack payloads and also can inject well-crafted noisy packets to contaminate the suspicious flow pool. To address these attacks, we propose SAS, a novel Semantics Aware Statistical algorithm for automatic signature generation. When SAS processes packets in a suspicious flow pool, it uses data flow analysis techniques to remove non-critical bytes. We then apply a Hidden Markov Model (HMM) to the refined data to generate state-transition-graph based signatures. To our best knowledge, this is the first work combining semantic analysis with statistical analysis to automatically generate

> worm signatures. Our experiments show that the proposed technique can accurately detect worms with concise signatures. Moreover, our results indicate that SAS is more robust to the byte distribution changes and noise injection attacks comparing to Polygraph and Hamsa.

The idea is to use HMMs to automatically extract signatures by first removing some of the "noise" so that the resulting signatures are stronger. It might be interesting to test this approach on metamorphic malware, which is a more challenging test case.

17. Here is the abstract from, A Robust and Fast Video Copy Detection System Using Content-Based Fingerprinting.

> A video copy detection system that is based on content fingerprinting and can be used for video indexing and copyright applications is proposed. The system relies on a fingerprint extraction algorithm followed by a fast approximate search algorithm. The fingerprint extraction algorithm extracts compact content-based signatures from special images constructed from the video. Each such image represents a short segment of the video and contains temporal as well as spatial information about the video segment. These images are denoted by temporally informative representative images. To find whether a query video (or a part of it) is copied from a video in a video database, the fingerprints of all the videos in the database are extracted and stored in advance. The search algorithm searches the stored fingerprints to find close enough matches for the fingerprints of the query video. The proposed fast approximate search algorithm facilitates the online application of the system to a large video database of tens of millions of fingerprints, so that a match (if it exists) is found in a few seconds. The proposed system is tested on a database of 200 videos in the presence of different types of distortions such as noise, changes in brightness/contrast, frame loss, shift, rotation, and time shift. It yields a high average true positive rate of 97.6% and a low average false positive rate of 1.0%. These results emphasize the robustness and discrimination properties of the proposed copy detection system. As security of a fingerprinting system is important for certain applications such as copyright protections, a secure version of the system is also presented.

A student recently completed a project where hidden Markov models (HMMs) were used to identify software (so as to aid in software piracy detection). For the relevant master's project, see the discussion for Problem 15, above. Such techniques (or similar) appear to be applicable to this "video copy detection" problem. So, the project would be to try various classification techniques for video/audio files (e.g., HMM-based) and compare the results. On the other hand, it might be possible to apply this (or similar) "content-based fingerprinting" method to malware detection and/or IDS.

18. Compare a bunch of different metamorphic detection techniques (HMM, $n$-gram similarity, chi-squared similarity, opcode graph similarity, cosine similarity/data mining, shortest common superstring, structural entropy, etc.). We would want to try combining scores to see if we can improve the results—standard methods for combining scores include Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) and various machine learning techniques. The first step would be to develop a substantial set of test data so that all methods could be tested on the same basis. This is related to Problem 5, above, but the goal here is to develop a (near) optimal scoring combination for a well-defined set of malware.

19. A paper that I recently reviewed (not yet published) talks about *in situ* (literally, "in place") attacks, that is, attacks that use code already present in a file. Here is the abstract:

    > Logical extraction of functional components from compiled programs is a new paradigm for functional component extraction that differs from the traditional physical approach. Using this new paradigm, the extracted functional components may be reused in situ; that is, without
    > first being separated from their original programs. Such in situ reuse is accomplished by usurping control of the program from which the functional component was logically extracted. Once control over the program's execution is attained, it may be driven to execute the code of the functional component contained therein. Several categories of logically extracted functional components have been identified, and the manner in which they may be reused varies. An implementation that allows for the programmatic in situ reuse of logically extracted functional components has been constructed, thereby providing proof of concept.

    The future work section includes the following:

    - Develop an automated method for discovering "functional components" that is, the bits of code that can be used in such an attack. It seems that HMMs could be very useful here.
    - Thread injection to obtain the functionality in 1. That is, inject a new thread into the process' execution context (in the paper, they use an impractical debugger script for proof of concept).

    A solid result here would be to construct real examples of metamorphic *in situ* malware. We have many tools from previous projects for analyzing metamorphic malware and these could be applied to your creation, giving us meaningful measures of the "quality" and "strength" of the resulting malware.

20. Twitter-based botnet to defeat the detection strategies in the paper "Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures", by Kartaltepe, et al. The abstract to this paper follows.

    > Botnets have become a major threat in cyberspace. In order to effectively combat botnets, we need to understand a botnet's Command-and-Control (C&C), which is challenging because C&C strategies and methods evolve rapidly. Very recently, botmasters have begun to exploit social network websites (e.g., Twitter.com) as their C&C infrastructures, which turns out to be quite stealthy because it is hard to distinguish the C&C activities from the normal social networking traffic. In this paper, we study the problem of using social networks as botnet C&C infrastructures. Treating as a starting point the current generation of social network-based botnet C&C, we envision the evolution of such C&C methods and explore social networks-based countermeasures.

    This is related to a previous master's project.

21. Technique to automatically recover information about a metamorphic generator from a set of metamorphic copies. This is essentially the idea in the paper given in Problem 9, but there are other possible approaches, such as HMM-based analysis.

22. Analyze the technique in the paper "Normalizing Metamorphic Malware Using Term Rewriting", and break it. The abstract is here:

    > Metamorphic malware — including certain viruses and worms — rewrite their code during propagation. This paper presents a method for normalizing multiple variants of metamorphic programs that perform their transformations using finite sets of instruction-sequence substitutions. The paper shows that the problem of constructing a normalizer can, in specific contexts, be formalized as a term rewriting problem. A general method is proposed for constructing normalizers. It involves modeling the metamorphic program's transformations as rewrite rules, and then modifying these rules to create a normalizing rule set. Casting the problem in terms of term rewriting exposes key challenges for constructing effective normalizers. In cases where the challenges cannot be met, approximations are proposed. The normalizer construction method is applied in a case study involving the virus called "W32.Evol". The results demonstrate that both the overall

approach and the approximation schemes may have practical use on realistic malware, and may thus have the potential to improve signature-based malware scanners.

23. Analysis of a classic cipher machine. There have been several previous master's projects of this type. For example, one student analyzed Sigaba (an American WWII-era cipher machine) and another analyzed Typex (a British variant of the Enigma machine). If you want to see either of these papers, send me an email.

24. Analysis of the Zodiac 340 cipher. This is an unsolved cipher produced by the infamous Zodiac killer. It is a longshot that you'd be able to break it, but there is plenty of scope for a good project related to this topic. For example, one previous student developed a general technique for attacking homophonic substitutions and applied it to the Zodiac 340. If you want to see that paper, send me an email.

25. HMM to detect metamorphic JavaScript virus. Here's a recent email I received:

    Dear Mark Stamp!

    I read many of your papers concerning metamorphic viruses, and as far as I remember, you only consider binary win32 metamorphic viruses.

    I wonder whether you can apply your technique also to other platforms - especially to script viruses. I suspect that this might have alot of practical relevance aswell.

    Recently I wrote a metamorphic JavaScript virus (`http://spth.virii.lu/MSCJS.txt` `http://spth.virii.lu/Transcriptase.rar` and `http://spth.virii.lu/transcript.pdf` for Peter Ferries analysis). The virus is still not detected by any anti-virus program, and just today i got an email that Peter Ferrie has given up writing the detection engine (for the moment?).

    From all that I understood your technique, I expect that a straight-forward adaptation to scripts is not possible, but I hope that you as expert can find a way. I would be very interested.

    kind regards!

    I haven't thought too much about it, but this looks like it could be a good project topic. And there's another email:

Dear Mark Stamp,

this is a wonderful idea, of course I dont mind. i am also available for questions concerning the codes.

just an idea, maybe a second code falls into that project aswell, namely my MicrophoneFever (`http://spth.virii.lu/Matlab.MicrophoneFever2.rar` and `http://spth.virii.lu/mifeve.pdf` the analysis by Peter Ferrie). This code is only detected my Microsoft, because Peter Ferrie who did the analysis found some weak points, the name "Not Mifeve" means "not my favorite", bceause he found it difficult :) )

i am looking forward to the potential results of that project! as i said, im always available for comments/questions.

kind regards!

I haven't looked at this second one yet.

It looks like it's possible to convert Javascript to bytecode, and that should fit in well with the general approach used in previous research.

> Yes, although webkit does generate an AST for parsing source, the interpreter (at least before the change to llint) operates on bytecodes. Calling jsc with the "-d" flag is enough to dump the bytecodes to the terminal. Or, the built-in debug() function will also dump out the bytecode format, it's defined in jsc.cpp.
>
> I just double-checked quickly, but it looks like the functionality to dump the bytecodes is still present in JavaScriptCore/bytecode/CodeBlock even after the change to llint, which means everything that I say (and that pertains to an older version) should still be relevant.
>
> For your convenience, I've attached my SecurityStackVerifier class, which does an abstract interpretation over the bytecodes. It focuses heavily on the instructions that affect control flow, and pretty much ignores everything else. But looking at it might give you some clues as to how to loop over the bytecodes when generating, inspecting, or comparing for virus signatures.
>
> I believe that you would make a call to the virus detector in JavaScriptCore/bytecompiler/BytecodeGenerator.cpp:ParserError BytecodeGenerator::generate() at least that's where we put the call to the stack verification (snippet below).

```
#ifndef NDEBUG
    m_codeBlock->setInstructionCount(m_codeBlock->instructions().size());

    if (s_dumpsGeneratedCode)
        m_codeBlock->dump(m_scopeChain->globalObject->globalExec());

    if (!m_codeBlock->verifySecurityStack(m_scopeChain->globalObject->globalExe
        m_codeBlock->dump(m_scopeChain->globalObject->globalExec());
#endif

bool CodeBlock::verifySecurityStack(ExecState* exec) const
{
    SecurityStackVerifier ssv(exec, this);
    return ssv.verify();
}
```

26. Hex-Rays (`https://www.hex-rays.com/products/decompiler/`) is a well-regarded decompiler. Find weaknesses in this decompiler, that is, find code constructs that confuse this decompiler. Ideally, we would like to develop a tool to automatically convert code into a form that is resistant to decompilation by Hex-Rays.

27. Code morphing consists of transposition, substitution, insertion, and (possibly) deletion. Can we use a Profile HMM (PHMM) to generate a code morphing "blueprint"? That is, we produce a PHMM that is then used to generate metamorphic copies of a given base file. Ideally, we should be able to apply the resulting PHMM-based morphing scheme to any program. But, initially, we could severely restrict the type of code that we consider (e.g., only "linear" code with no loops or conditionals).

28. Here's the abstract from a soon-to-be published paper:

    > Fialka M-125 (sometimes called the "Russian Enigma") is an electro-mechanical rotor cipher machine used during the Cold War. The designers of this cipher eliminated the known weaknesses of Enigma. In this paper, we summarize the main principle of the Fialka algorithm from public sources. Moreover, we introduce a mathematical model of the Fialka cipher, and we analyze the effect of blocking pin settings on the cipher's period.

    The analysis in this paper raises some interesting possibilities. For example, there might possibly be non-obvious ways to weaken the cipher by careful choice of rotor wirings and numbers of "blocking pins" per rotor. In any case, a good project topic would be to cryptanalyze Fialka from a modern perspective. Previous student projects involved the cryptanalysis of Sigaba (an American cipher machine developed in the 1940s) and Typex (essentially, the British version of Enigma).

29. A score based on byte distributions in files. This could be modeled on the "structural entropy" score in Dona's paper (and Jared's paper), but using, say, a simple $\chi^2$ statistic in place of an entropy or compression score.

30. Experiment with metamorphic techniques designed to defeat function call graph similarity (like that analyzed in Prasad's paper).

31. Develop and analyze a score based on flow graph (and/or other graphs, such as function call graph). Use graph matching approximation algorithms to compute the score between two such graphs—see, for example, The approximate graph matching problem, by Wang, Zhang, and Chirn.

32. Use wavelet compression as the basis for a malware score. This would be related to Dona's paper and Jared's work (sort of a combination of their approaches). Wavelet compression consists of computing a wavelet transform then compressing the coefficients. The compression can be lossy of lossless. The simplest example of lossy approach would be to simply discard the coefficients below a certain threshold. The topic of wavelet compression (in a different context) is discussed here: `http://cva.stanford.edu/classes/ee482s/homework/hw1.pdf` An introduction to wavelets can be found here: `http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html` Another good introduction is here: `http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html`

33. Develop and analyze a wavelet-based score analogous to that in Dona's and Jared's papers, but based on HMM scores. That is, compute HMM scores over small windows of data, then apply a wavelet analysis to detect high-scoring sections of the code. This could be tested on opcode sequences, but it would be most interesting if applied to the .exe files themselves (specifically, the .text section), which is closer to the approach in the previous work. The idea is that if we train an HMM on virus files that do not contain any dead code, then the model would score well on sections that contain actual virus code. The wavelet analysis could be used to extract these segments of the file that score well agains the model. Then these extracted segments could, at the least, provide an estimate for the amount of dead code in a (known/suspected) virus file. Its a bit trickier to derive an actual score for an unknown file, but that should also be possible.

34. Metamorphic detection using Conditional Random Fields (CRF).

35. Metamorphic detection using Finite State Transducers (FST). One approach would be to train multiple HMMs, then convert each to an FST, which is an approximation to

the original HMM—see the paper, Finite State Transducers Approximating Hidden Markov Models, by A. Kempe. Once in the form of an FST, detection should be faster, it may be possible to manually correct for errors in the HMM, and there is an algorithm for composing FSTs. The composition of FSTs could be interesting, since it would allow us to combine the multiple HMMs into one (possibly, significantly faster) detection pass. If the original HMMs were trained on different aspects of a given metamorphic family, then the combined model might be much stronger than any of the individual models. On the other hand, if the original HMMs were trained on different metamorphic families, then the composition might enable the simultaneous detection of multiple types of malware, which might be much, much faster than detection each independently.

36. Metamorphic detection using Position Weight Matrices (PWM) and higher order PWMs (see Khuri's recent master's project), or possibly "Message Passing and Sinkhorn Balancing Algorithms" (see Lakshmi Ananthagopal's project).

37. Work on a followup to Chinmayee's project. Specifically, try to apply clustering to malware detection, instead of just to classify. Also, use more advanced clustering algorithms, including Expectation Maximization (EM).

38. Compare hill climb to HMM with (millions of) random starts. The simple substitution cipher would be a good test case. This paper discusses the use of an HMM with (millions of) random restarts as a way to attack a homophonic substitution cipher: `http://www.cs.berkeley.edu/~tberg/papers/emnlp2013.pdf` The simple substitution attack is easier. This approach would also work well on the WWII Japanese Purple cipher, which would make another interesting test case.

39. Design an authentication protocol for NFC and use Nikki's framework to test it. Here is something similar for RFID: `http://onlinelibrary.wiley.com/doi/10.1002/sec.965/abstract` Another possible path to pursue is the use of crypto in NFC, such as is discussed here for RFID: `http://link.springer.com/article/10.1007/s10207-014-0236-y`

40. In `http://spth.virii.lu/transcript.pdf` it is claimed (regarding the Transcriptase metamorphic JavaScript malware) that

    > The assumption is that detection of a metamorphic virus is more difficult than detection of an ordinary virus for an anti-virus engine. While this is certainly true, the act of making a virus metamorphic introduces so much 'noise' that, in a sense, detection is not always as difficult as the virus writer intended. Since the resulting code contains so much obfuscation, it rarely resembles the code of a regular program. This allows us to find all

> kinds of artefacts [sic] which attract our attention, and that in turn allows us to spend more time scanning, with no impact on ordinary users who tend not to have such samples. We can even take this further—hundreds of hours of the virus writer's work can be undone in a matter of a few hours by an anti-virus researcher. The existing metamorphic viruses have been detected in a matter of days (once the time was devoted to writing a detection, of course), in contrast to the months of work put in by the virus writer. Given that, you have to wonder why the virus writers bother.

Design and implement a strong metamorphic generator that does not produce so much "noise". That is, the generator should morph the code without doing things that would not be expected of benign code.

41. We have a short message that was supposedly encrypted with the Typex cipher during WWII, but has never been decrypted. We have started to do some work trying to break it, but there is room for a master's project. You would work on a fast implementation of the attack, and analyze ways to reduce the search space for the attack.

42. Use an HMM to recover unknown rotor in Typex (or Enigma). The idea would be to do something analogous to the hill climb attack in Kelly Chang's paper, but using an HMM. To start, we could try to recover a single unknown rotor in a much simplified rotor machine. Have to think more about how to do this, but it should work, since Kelly's attack was based on Jakobsen's algorithm.

43. A paper that I've just recently reviewed analyzes several different graph techniques for malware detection This work is done in the context of "zero day" malware, i.e., malware that has not previously been seen. Here is the abstract from that paper:

    > Due to the unavailability of signatures for previously unknown malware, zero-day malware detection schemes typically rely on analyzing program behavior. Prior behavior based zero day malware detection schemes are either easily evadable by obfuscation or are very inefficient in terms of storage space and detection time. In this paper, we propose GZero, a graph theoretic approach fast and accurate zero-day malware detection at end hosts. The key advantage of GZero over existing schemes is that it is inherently more robust to malware obfuscation, while being efficient in terms of both storage space and detection time. We conducted experiments on a large set of both benign software and malware. Our results show that GZero achieves 100% detection rate and a false positive rate of less than 2%, with less than 1 second of average scan time per program and is robust

> to obfuscation attacks. Due to its low overheads, GZero can complement existing malware detection solutions at end hosts.

Similar techniques can be applied to the metamorphic detection problem. In fact, I believe the metamorphic case provides a better testing environment for this work than the (ill-defined) "zero day" malware discussed in the paper. Other twists on this theme could include analyzing features based on clustering results, rather than those based directly of graphs. Other related ideas include the use of, say, HMM analysis of the features (graph and/or cluster).

44. I've recently reviewed a (not very good) paper that analyzed API call sequences as a way to detect rootkits in Windows. They constructed graphs based on the system calls and compared the graphs by matching the "longest common subsequence" between the graphs. This could be explored in the context of metamorphic malware. Other related measures could also be analyzed, such as the "longest common superstring". We would want to analyze the effectiveness of such approaches, and consider techniques that a malware writer could use to defeat such detection strategies.

45. I've reviewed a paper that discusses a system based on using agents to send info about detected malware to all other hosts on a network. That is, if one host detects malware, it informs all other hosts (and a central database) of the attack. Here's the abstract:

    > Nowadays malware has become a major threat to computers and information systems on the internet. With the spread of internet and network-based services, malware has spread widely and thus there is an increasing need for an automatic security system. To have an automated security system and restrict the spread of malware in the network, it has become essential to share information about any detected malware with network partners and peers. Our system uses signature-based methods for automatic detection and analysis of malware. In this paper we propose an automated security system for sharing malware information found on users' computers to develop advice to protect other computers. Our system has the ability to automatically analyze the detected malware on clients' computers. The main agent shares malware information with other users in the same network. Alerting other users, by sharing information about malware detected in any computer in a client server network environment, can develop advice to protect other computers in the network. The results from an implementation of the proposed system show that our approach is effective in analyzing detected malware in the automated security system.

I'm not convinced that this method as described is valuable in a typical networked situation. Usually, all hosts would be doing the same scanning as the detecting host. In addition, the use of agents opens the door to additional attacks, and this additional risk does not seem to come with any significant potential benefit.

Possible related projects would be to develop an agent-based system for an environment where such a system has a clearer benefit. For example, if the detection technique is extremely costly and the work can be distributed among the hosts, then such an agent-based approach would make sense. Or, in cases where the hosts are severely resource-limited (and, again, the detection workload can be distributed), such an approach would be reasonable. Another line of study would be to look for attacks on such a communication system. For example, the attacker could create an evil agent that communicates false attack info to the hosts.

46. Use the simple substitution-column transposition attack in Jeffrey Yi's paper as a score to detect metamorphic malware. This would be very similar to Gayathri's paper: `http://link.springer.com/article/10.1007/s11416-013-0184-5`. The biggest challenge here would be to find a dataset (or ideally, datasets) that show poor results on the simple substitution score and (hopefully) better results for this score.

47. Test the HMM with millions of random restarts (see topic 38, above) as a way to score malware.

48. I've recently reviewed a paper that discusses "malware biodiversity". Here is the abstract:

> Malware is constantly changing and is released very rapidly, necessarily to remain effective in the changing computer landscape. It then stands to reason that malware samples will be related to each other; studies that indicate that malware samples are similar often base that determination on common behavior or code. Given, then, that malware samples are developed based on existing samples, we can see that some code fragments, behavior, and techniques may be influencing more development than others. We propose a method by which we can determine the extent that previously released malware samples are influencing the development of new samples. Our method allows us to examine the way that malware changes over time, allowing us to look at trends in the changing malware landscape. This method, which involves a historical study of malware, can then be extended to investigate specific behaviors or code fragments. Our method shows that in general, over 64% of malware samples that we analyzed are contributing to the biodiversity of the malware ecosystem and influencing new malware development.

That is, the authors examine code fragments to identify similarity between different malware, and use this similarity to determine connections between malware. It makes some sense to look for commonality (and differences) by looking directly at the code, but one possible weakness is that minor variations in code may look very different by such a measure when, in reality, they are actually very similar. In previous research, students have developed a wide variety of software similarity measures (statistical, structural, graph based, etc.), many of which are fairly robust with respect to slight variations in the code. I'd like to see a project (or projects) where we look at the evolution of malware, based on "biodiversity", as measured by some of the scores developed in previous masters projects. We could train models based on well-defined malware families that are of fairly recent origin. Then, we could use these models to score historical malware samples with the goal of tracing the evolution (as measured by a particular similarity score or combination of scores) of the modern families. The difficulties in this project/projects include collecting a good set of malware samples for study, and the analysis of the resulting scores may be a bit tricky. However, we have many scores available that have been studied in the context of malware detection, so there would be no need to develop or analyze new scoring techniques for this research.

49. The following is the abstract of a not-yet-published paper.

    > Malicious Java applets are widely used to gain access to remote computer systems by installing further malicious software. In this work, we present HoneyAgent which allows for the dynamic analysis of Java applets, bypassing common obfuscation techniques. This enables security researchers to quickly comprehend the functionality of an examined applet and to unveil malicious behavior. In order to trace the behavior of a sample as far as possible, HoneyAgent is further able to simulate various vulnerabilities allowing analysts for example to identify the malware that should finally be installed by the applet. In our evaluation, we show that HoneyAgent is able to reliably detect malicious applets used by common exploit kits with no false positives. By using a combination of heuristics as well as signatures applied to observed method invocations, HoneyAgent is further able to identify exploited CVEs in many cases.

    A good project would consists of breaking this detector. That is, develop a Java obfuscator that can modify malicious code so that it is not detected by "HoneyAgent".

50. A paper I recently reviewed (actually, the same paper that is mentioned in number 49) makes the following claim.

> Many exploits used by exploit kits target the Java Runtime Environment. Recent reports claim that Java alone made up for 91 percent of web based exploits in 2013 and more recent exploit kits, e.g. the Neutrino exploit kit, even solely rely on Java exploits. To trigger a Java exploit, a malicious Java applet is embedded into a web page where it will be downloaded and executed by the victim's web browser. These applets are often strongly obfuscated to prevent signature-based detection, as used by most antivirus solutions.

It would be interesting to analyze a set of obfuscated malicious applets to determine whether the scores developed in previous projects (e.g., HMM-based score) can successfully detect them. That is, we would like to analyze static detection techniques as opposed to the dynamic analysis considered by the authors of this paper.

51. Detecting encrypted malware using HMMs, as compared to detection using a basic entropy measure, such as that discussed here: `https://www.cs.jhu.edu/~astubble/600.412/s-c-papers/keys2.pdf`. More generally, perhaps use HMM scores over short sequences and then apply transform techniques to partition the file into code, data, encrypted, etc.

52. Here's the abstract from the paper, "MysteryChecker: Unpredictable Attestation to Detect Repackaged Malicious Applications in Android".

> The number of malicious applications, sometimes known as malapps, in Android smartphones has increased significantly in recent years. Malapp writers abuse repackaging techniques to rebuild applications with code changes. Existing anti-malware applications do not successfully defeat or defend against the repackaged malapps due to numerous variants. Software-based attestation approaches widely used in a resource-constrained environment have been developed to detect code changes of software with low resource consumption. In this paper, we propose a novel software-based attestation approach, called MysteryChecker, leveraging an unpredictable attestation algorithm. For its unpredictable attestation, MysteryChecker applies the concept of code obfuscation, which changes the syntax in order to avoid code analysis by adversaries. More precisely, unpredictable attestation is achieved by chaining randomly selected crypto functions. A verifier sends a randomly generated attestation module, and the target application must reply with a correct response using the attestation module. Also, the target application periodically receives a new module that contains a different attestation algorithm. Thus, even if the attacker analyzes the attestation module, the target application replaces the existing attestation module with a new one and the analysis done by the attacker becomes invalid.

> Experimental results show that MysteryChecker is completely able to detect known and unknown variants of repackaged malapps, while existing anti-malware applications only partially detect the variants.

We could use metamorphic techniques to derive an analogous "unpredictable attestation" technique, which might be better than the approach considered in the paper. A good project would consist of studying the approach used in this paper and finding weaknesses with it. Then we would develop our metamorphic version and show that it is stronger. Using previously developed metamorphic analysis and/or software piracy detection techniques, we could measure the improvement offered by our approach.

53. Here is the abstract from a recently-published paper titled "CrowdSource: Automated Inference of High Level Malware Functionality from Low-Level Symbols Using a Crowd Trained Machine Learning Model".

> A common first step malware analysts take in understanding a suspicious or malicious binary is to extract printable byte sequences (strings) from the binary and then inspect any technical symbols and natural language evident in these data, thereby making inferences about the program's high-level functionality and purpose.
>
> Given the enormous volume of new malware binaries appearing on the Internet, it would be useful to automate this "first pass" analysis of malware binary programs. A challenge, however, is to replicate the expert human intuition required to interpret these data, which involves knowledge of the meaning of many thousands of arcane API calls, protocol symbols, and natural language word sequences.
>
> In this paper we propose CrowdSource, a statistical natural language processing system designed to make rapid inferences about malware functionality based on strings data. CrowdSource "learns" a mapping between low-level language and high-level software functionality by leveraging millions of web technical documents from StackExchange, a popular network of technical question and answer sites, as a knowledge base, using this mapping to infer malware capabilities.
>
> While CrowdSource cannot replace human reverse engineers or solve the problem of unpacking and deobfuscating malware, we demonstrate that the CrowdSource approach can accelerate the malware analysis process after malware has been unpacked by making "low hanging fruit" inferences, and by allowing for the rapid generation of new capability detectors as analyst users require them.
>
> This paper describes the CrowdSource approach and provides an evaluation of the system's accuracy and performance, demonstrating that it can detect

> at least 14 high-level malware capabilities in unpacked malware binaries with reasonable accuracy and at a rate of tens of thousands of binaries per day on commodity hardware.

We could do a similar sort of analysis, but based on extracted opcodes, instead of "printable byte sequences". That is, we would want to train models to detect high-level information from low level (e.g., opcode) information. By doing so, we might be able identify malware-like behavior from very low level (static) analysis.

54. Here is the abstract from a recent paper titled "Codescanner: Detecting (Hidden) x86/x64 Code in Arbitrary Files".

> Disassembly is indispensable for the proper analysis of malware. However, a common problem concerning the x86/x64 architecture is that disassemblers produce partially incorrect results. This is used by malware authors who nowadays routinely generate binaries with anti-disassembly measures. In this paper, we derive general constraints on x86 code which are not based on disassembly but on byte level. Based on these constraints we develop a set of classifiers able to locate code in any kind of files. Operating on byte level, our approach is independent of assembly semantics. Our evaluation shows that we are able to precisely locate code and provide anti-disassembly resistance independent of the operating system or compiler. Our tool can be used to detect the code sections of malware, improve code coverage for disassemblers, detect hidden code in files and in memory, or identify malware with anti-disassembly techniques.

The point here is that it is often useful to detect code hidden within a file, without having to try to disassemble it. There is another interesting case that is explicitly not considered by the authors. Often, in malware files—specifically, encrypted or polymorphic malware—encrypted code is present. It would be useful to detect such encrypted code and, in fact, the presence of such code could serve as a strong heuristic that the code is actually malware. The encryption used in malware is often weak, such as XOR with a fixed byte value. Such an encryption scheme is equivalent to a simple substitution cipher. Therefore, we could use a well-known fast and efficient method for breaking a simple substitution cipher (i.e., Jakobsen's Algorithm) to help us detect encrypted malware. That is, if we find a "data section" that looks like code after decrypting, then it is probably encrypted/polymorphic malware. For this project, the approach would essentially be a combination of the methods used in the paper mentioned above and Gayathri's paper, which can be found here: `http://link.springer.com/article/10.1007/s11416-013-0184-5`

55. Here's the abstract from a recent paper titled, "Bacterial Quorum Sensing for Coordination of Targeted Malware".

    > Bacterial Quorum Sensing is a process that bacteria use to determine their local population density. Based on this determination, individual bacterial cells may alter their survival strategies to those strategies which benefit the cell the most. For example, bacteria utilize quorum sensing to determine if the cell would benefit more from either asocial or social strategies. Alone, a single cell is vulnerable, but in a community they represent a threat capable of overwhelming a host's immune system. Most importantly, most quorum sensing approaches use commonly-encountered chemicals for sensing; due to their ubiquity, these quorum signals do not become useful for determining if an object is a bacterium; rather, they speak to the local population density.
    >
    > Similarly, malware has demonstrated a variety of techniques to communicate and to evade detection, and like bacteria, survival strategies can also depend on population density. As such, malware could utilize the bacterial quorum sensing system as a method of communication which has the potential to allow targeted malware to communicate and coordinate activities. Furthermore, inspired by bacterial quorum sensing, malware could use signals that are already common in the computing environment in a way that does not provide actionable remediation intelligence to network defenders. Thus, the use of a bacterial quorum sensing mechanism instead of another distributed algorithm allows the malware to leverage self-organizing properties that are based to the number of infected hosts on a network without exposing individually infected hosts to targeted remediation. This paper demonstrates and implements a digital version of the quorum sensing system through a timing covert channel, and uses statistical tests to determine if a signal is present. We argue that just as for bacteria, the digital quorum sensing signal is not useful for determining if a particular host is infected; as such, it is an attractive choice for malware authors.

    The authors claim that you can create a covert channel for communicating with malware (e.g., a botnet) that mimics the process of "bacterial quorum sensing". It would be interesting to implement this approach on a simulated botnet and compare it to other more standard covert channel techniques, as well as other biology-inspired covert channels (e.g., ant colonies). Previously, students have developed and analyzed a botnet that uses Twitter for its command and control (`http://www.mecs-press.org/ijcnis/ijcnis-v5-n6/IJCNIS-V5-N6-2.pdf`) and a simulator based on that botnet might be a good one for this research. Also, Ismeet is currently developing a simulator for this botnet as part of her CS 298 project, and we could use that as the starting point for this project.

56. Here is the abstract from a recent paper titled, "Host-Based Code Injection Attacks: A Popular Technique Used By Malware".

    > Common goals of malware authors are detection avoidance and gathering of critical information. There exist numerous techniques that help these actors to reach their goals. One especially popular technique is the Host-Based Code Injection Attack (HBCIA). According to our research 63.94% out of a malware set of 162850 samples use HBCIAs. The act of locally copying malicious code into a foreign process space and subsequently executing it is called a Host-Based Code Injection Attack.
    >
    > In this paper, we define HBCIAs and introduce a taxonomy for HBCIA algorithms. We show that a HBCIA algorithm can be broken down into three steps. In total there are four classes of HBCIA algorithms. Then we examine a huge set of malware samples and estimate the prevalence of HBCIA-employing malware and their target process distribution. Moreover, we analyse Intrusion Prevention System data and show that HBCIA-employing malware prefers network-related processes for its network communication.
    >
    > To the best of our knowledge, we are the first to thoroughly describe and formalize this phenomenon and give an estimation of its prevalence. Thus, we build a solid foundation for future work on this topic.

    The particular problem studied here does not seem all that interesting, but the related idea of detecting certain common behavior in a (large) set of malware samples is a good one. We could use machine learning techniques to classify small sections of code (ideally, based on common behavior) and measure their prevalence in malware, as compared to benign files. The strongest such "features" could then be used for detection, where we might have a good chance of detecting new malware since it would likely have similar features. To accomplish this, we might want to use a technique similar to that mentioned in topic 53, since such an approach could give us a better view of actual "behavior" as opposed to just statistical similarity. Or, we might keep it simple and use standard machine learning strategies, such as HMMs. In any case, one significant difficulty is likely to be finding short segments of relevant code within a file. For this, some ideas similar to those used to segment files in `http://link.springer.com/article/10.1007/s11416-013-0185-4` (which is based on transform techniques) might be useful.

57. Here is the abstract from a really fascinating paper titled, "AirHopper: Bridging the Air-Gap between Isolated Networks and Mobile Phones using Radio Frequencies".

    > Information is the most critical asset of modern organizations, and accordingly coveted by adversaries. When highly sensitive data is involved, an

> organization may resort to air-gap isolation, in which there is no networking connection between the inner network and the external world. While infiltrating an air-gapped network has been proven feasible in recent years (e.g., Stuxnet), data exfiltration from an air-gapped network is still considered to be one of the most challenging phases of an advanced cyber-attack.
>
> In this paper we present "AirHopper", a bifurcated malware that bridges the air-gap between an isolated network and nearby infected mobile phones using FM signals. While it is known that software can intentionally create radio emissions from a video display unit, this is the first time that mobile phones are considered in an attack model as the intended receivers of maliciously crafted radio signals. We examine the attack model and its limitations, and discuss implementation considerations such as stealth and modulation methods. Finally, we evaluate AirHopper and demonstrate how textual and binary data can be exfiltrated from physically isolated computer to mobile phones at a distance of 1-7 meters, with effective bandwidth of 13-60 Bps (Bytes per second).

The idea is that the authors are able to extract data from an air-gapped network by simply having an infected cell phone nearby. I don't have any immediate ideas for a related project, but if you find this especially interesting, we can discuss it. I'm sure there are some good possibilities here.

58. In a somewhat similar vein as topic 57, here's the abstract from a recently published paper titled, "Exfiltrating data from Android devices".

> Modern mobile devices have security capabilities built into the native operating system, which are generally designed to ensure the security of personal or corporate data stored on the device, both at rest and in transit. In recent times, there has been interest from researchers and governments in securing as well as exfiltrating data stored on such devices (e.g. the high profile PRISM program involving the US Government). In this paper, we propose an adversary model for Android covert data exfiltration, and demonstrate how it can be used to construct a mobile data exfiltration technique (MDET) to covertly exfiltrate data from Android devices. Two proof-of-concepts were implemented to demonstrate the feasibility of exfiltrating data via SMS and inaudible audio transmission using standard mobile devices.

Last year, Nikki did an excellent project involving NFS on Android, so she's familiar with a lot of the topics discussed in this paper. After seeing this paper, she made the following comments.

> Skimming through this has made me nostalgic for my project… they hit all the same stupid Android problems I did, like how to get an app installed on an Android phone without someone really agreeing to it.
>
> Their Smali injection writeup could be useful for a totally different project. I feel like someone's project last year hit on how hard it is to recompile decompiled Java, right? One of the metamorphoic malware ones. Looks like they have some good detail in the writeup here that could be of practical value to some poor master's student.
>
> The inaudible tones thing is a cool idea, very similar to a paper I read for MALCON (AirHopper I think). It could be fun to do a similar app for iOS (and you know people always go nuts when there's any sort of successful security breach with iPhones). Or, to poke at their claim that the only way to prevent the audio exfiltration is to add a permission to using the audio output.
>
> Or, to combine their inaudible exfiltration with some sort of further relay of the data via other infected phones. It seems like their proof of concept app, if no one malicious is standing within 3 meters, it doesn't really matter that the data's getting exfiltrated, no one's receiving it. But, if there were other infected phones nearby to relay from, you might eventually get it somewhere useful. Maybe borrow something from delay tolerant networks and have the infected phone hold the data til a malicious phone comes in range and sends a beacon signal or something… anyway point being, there's something that can be done about the range limitation here. (I don't know what they're complaining about anyway, 3m is a lot better than 4 cm.)
>
> The SMS idea is, as they admit themselves, pretty unsubtle. And they only even talk about one or two of the least subtle points. Imagine their app running on someone's phone when they're (for instance) out on a hike. Taking a ton of pictures—app's trying to exfiltrate them constantly—but there's little or no cellular signal - so there'd be a ton of "message failed" notifications showing up for messages the user had no recollection of sending. Not subtle. Could be made more subtle for sure (check signal before sending, combine multiple messages, etc etc) but still IMO not as cool as the inaudible tones thing.

If any of this looks interesting, let me know and I can send you a copy of the paper.

59. Use dynamic "software birthmark" analysis techniques for piracy detection. We would consider the special case where the software pirate modifies the existing code in an attempt to hide the fact that it's pirated. This has quite a bit of overlap with

techniques used for metamorphic malware detection. Previous research in this area includes Hardik's paper and Shabana's project, both of which used static analysis. Here, we would want to use dynamic analysis and (hopefully) show that we can improve on the previous results. Dynamic analysis techniques are being considered (in the related context of malware detection) in Anusha's and Swapna's current projects. If you want more info, let me know and I can send you Hardik's and Shabana's papers. (Note to self: This looks like a good topic to publish in the journal *Software: Practice and Experience*).

60. There is a recently published paper titled, Identifying Android malware using dynamically obtained features, that can be found here: `http://link.springer.com/article/10.1007/s11416-014-0226-7`. A good project would consist of testing static analysis techniques (HMM score and others) on the same dataset and comparing your results to those in this paper.

61. The solved Zodiac 408 cipher was a homophonic substitution cipher. A homophonic substitution is like a simple substitution, except that more than one ciphertext symbol can correspond to a single plaintext symbol. There is another ciphertext known as the Zodiac 340 which looks similar to the Zodiac 408. However, the Zodiac 340 remains unsolved. Recently, a student (Amrapali) developed a nested hill-climb attack for a homophonic substitution—the published paper is here `http://www.tandfonline.com/doi/full/10.1080/01611194.2013.797041#.UmaEMyithdM` and an extended version of the paper is available here `http://www.cs.sjsu.edu/faculty/stamp/RUA/homophonic.pdf` This hill climb attack is based on Jakobsen's algorithm, which is a fast method for solving a simple substitution.

    Another homophonic substitution solver, known as zkdecrypto, is available here `http://code.google.com/p/zkdecrypto/`. The zkdecrypto algorithm seems to be somewhat similar to Amrapali's algorithm, with two notable exceptions. First, zkdecrypto does not do a nested hill climb. In Amrapali's approach, the outer hill climb consists of testing different distributions of letters (e.g., test different numbers of ciphertext symbols mapping to, say, plaintext E). Instead, zkdecrypto makes an initial guess for the distribution of letters and that never changes. Second, zkdecrypto uses bigram, trigrams, up to pentagrams, with the higher order $n$-grams weighted more. As in Jakobsen's algorithm, Amrapali only use bigrams.

    Can we develop a variant of Jakobsen's algorithm that enables us to score higher-order $n$-grams, while still being reasonably efficient? If so, how does this compare to Jakobsen's algorithm for a simple substitution? And how much does it help in the homophonic substitution? And, can we apply it in the case of a combination cipher that consists of a simple substitution with a column transposition.

62. Simple substitution (or SSCT) with random restarts? How much can we improve

Jakobsen's (or Yi's) algorithm by using multiple random restarts? How about in the case where these are applied to malware scoring?

63. This paper discusses a method for stealing keys from a PC using a radio: `http://www.tau.ac.il/~tromer/radioexp/` Suppose that malware has been installed on a PC. Can we use a similar radio technique to exfiltrate important data?

64. Detecting image spam is a difficult challenge. Here is the abstract from a recent (as yet unpublished) paper on the topic.

    > The evasion techniques used by Image spams impose new challenges for e-mail spam filters. Effectual image spam detection requires selection of discriminative image features and suitable classification scheme. Existing research on image spam detection utilizes only visual features such as color, appearance, shape and texture, while no effort is made to employ statistical noise features. Further, most image spam classification schemes assume existence of clear cut demarcation between extracted features from genuine image and image spam dataset. We attempt to solve these issues; by proposing a novel server side solution called F-ISDS (Fuzzy Inference System based Image Spam Detection Scheme). F-ISDS considers statistical noise features along with the standard image features and meta-data features. F-ISDS employs dimensionality reduction using Principal Component Analysis (PCA) to map selected set of $n$ features into a set of $m$ principal components. Based on the selected significant principal components, input/output membership functions and rules are designed for Fuzzy Inference System (FIS) classifier. FIS provides a computationally simple and an intuitive means of performing the image spam detection. Email server can tag email with this knowledge so that client can take decision as per the local policy. Further, a Linear Regression Analysis is used to model the relationship between selected principal components and extracted features for classification phase. Experimental results confirm the efficacy of the proposed solution.

    Additional relevant resources include the following papers:

    - G. Yan, A. Choudhary, G. Hua. A comprehensive server to client side approach to image spam detection. Proc. *IEEE Transactions on Information Forensics and Security*, Vol. 5, No. 4, December 2010.
    - Z. M. Win, N. Aye. Detecting Image Spam Based on File Properties, Histogram and Hough Transform *Proc. of Journal of Advances in Computer Networks*, Vol. 2, No. 4, December 2014.

For this project, we would apply some of the techniques that have previously been developed to detect and/or classify malware to the image spam detection problem. Techniques such as HMMs, SVMs, clustering, and so on, would all seem to be applicable to this problem.

65. Here is the abstract of an unpublished (as of now) paper that analyzes a malware detection (and/or classification) score based on system call graphs.

> In this paper we present a graph-based model that, utilizing relations between groups of System-calls, detects whether an unknown software sample is malicious or benign, and classifies a malicious software to one of a set of known malware families. More precisely, we utilize the system-call dependency graphs (or, for short, ScD graphs), obtained by traces captured through taint analysis, and a set of various similarity metrics in order to perform the processes of detection and classification. We empower our model against strong mutations applying our detection and classification techniques on a weighted directed graph, namely Group Relation Graph, or Gr-graph for short, resulting from ScD graph after grouping disjoint subsets of its vertices. For the detection process, we utilize the Euclidean distance and propose the $\Delta$-similarity metric, utilizing the in-degree and out-degree of Gr-graph's nodes along with their corresponding weights, distinguishing thus graph-representations of malicious and benign samples. Moreover, for the process of classification, we propose the SaMe-similarity and NP-similarity metrics that utilize quantitative, relational and qualitative characteristics of Gr-graphs in order to index an unknown malware sample to a known malware family. Finally, we evaluate our model for malware detection and classification showing its potentials against malicious software measuring its detection rates and classification accuracy and discuss our results together with those achieved from relevant models.

A good project would consist of implementing a system call-graph score (somewhat similar to the one in this paper) and testing it against several of the metamorphic families that have been used in previous research. We would then be able to compare our new score to previously developed scores. We would also want to look at ways to try to break the score.

66. Here's the abstract from a recent (as yet unpublished) paper on the topic of click fraud in the context of malware.

> Advertising fraud, particularly click-fraud, is a growing concern for the online advertising industry. The use of clickbots, malware that automatically

> clicks on ads, to generate fraudulent traffic has steadily increased over the last years. While the security industry has focused on detecting malicious binaries associated with clickbots, a better understanding of how fraudsters operate within the ad ecosystem is needed to help design appropriate defense strategies and reduce the impact on affected businesses.
>
> This paper presents a new methodology to study ad-fraud malware. As an example, we provide a detailed dissection of the advertising fraud scheme employed by Boaxxe, a malware specializing in click fraud. We focus solely on the automated mode of Boaxxe, in which click fraud is performed without requiring any user action. The infected computer passes through a long chain of redirections interconnecting various actors, before ending on the actual ads. By monitoring the network traces from bots over several months, we automatically reconstructed the redirection chains and mapped the actors involved in the ad fraud scheme. We successfully characterized the activity of Boaxxe, defined its infrastructure and identified the manner in which a known media group is involved. We assert that this work demonstrates the benefits of having an automated methodology to conduct detailed analysis on the behavior of clickbots.

We could do a similar analysis with some other click-fraud related malware. Collecting the data would probably be the most challenging part of this project.

67. In some applications, we need to train an HMM many, many times using different initial starting points—in some cases, millions of random restarts must be tested. Examples of problems where such a technique has been successfully applied include using an HMM to break a simple substitution cipher, homophonic substitution cipher, or various combination ciphers, especially in cases where the amount of ciphertext is limited. And, if it could be made more efficient, such a technique would have potential for use in many other applications, since it would tend to improve the results of any HMM.

    The goal of this project is to see if we can get similar results as those obtained via millions of random restarts, but with a more efficient process. One possibility is to simply apply random modifications to the $B$ matrix, once the model has ceased to effectively improve. Perhaps this could be done as part of a hill climb, in the sense, that we modify the $B$ matrix until the model score improves, then we continue with the Baum-Welch algorithm from that point. Another possibility is to combine an HMM with a genetic algorithm (GA), call it an HMGA. After training an HMM, we could apply a step of GA to the $B$ matrix (using a crossover function consisting of swapping rows of $B$, for example), and then apply more iterations of Baum-Welch. If the model score does not improve, we can try a different GA modification to $B$ and so on. Combining HMMs and GAs has been considered previously, but I have not seen any work that is exactly like what is being proposed here. Yet another possibility is to apply something analogous to a boosting strategy (see, for example, AdaBoost) to the $B$ matrix at various intervals. In any case, the goal is to prevent the hill climb from getting "stuck" on sub-optimal hills for too long.

68. New-and-improved SSCT score: To determine column perm before simple substitution hill climb, do a hill climb on the column perm, using "digraph coincidence index" (like IC, but based on digraphs) as the score.

69. HMM with random restarts for malware detection score. Test this approach in cases where data is limited—want 3-d graphs of AUC as a function of data size (i.e., number of malware files in training set) and number of random restarts. And, do this for several different malware families.

70. Covert botnet in Android. Extend Sukanya's work so that data can be silently infiltrated as well as exfiltrated, and then use this to build a botnet.

71. Realistic scan based on extracted features using machine learning. Extract features over large set of exe files (like everything on a typical Windows system) with just a small number of family malware samples included. Determine how well we can do at scanning.

72. "Binning" for malware detection. That is, use an approach analogous to Galton's binning (which was used in pre-computer times for fingerprint matching) for malware detection. Binary-search type of approach.

73. Invariants of metamorphic malware — automatically extract defining characteristics and/or features from a given metamorphic family.

74. Here is part of the (edited) abstract from a paper I recently reviewed.

    > Techniques for malware analysis are generally based on two-way classification of applications behavior, i.e., malicious (harmful) or benign (not harmful). In many cases, the two-way treatment of applications behavior may not be effective. For instance, a malicious application may occasionally behave like benign in order to deceive the analysis engine thereby leading to ambiguous information which may not be sufficient for the sake of precise classification. Intuitively, the available information in such cases is not sufficiently clear and convincing to make immediate and accurate two-way classification decisions. Perhaps a better and more useful approach would be to delay the classification of difficult cases in the hope that future (or additional) information will make their classification more evident. In this

> paper, we investigate a three-way decision making approach involving an option of delay or deferment which is exercised whenever the available information is not adequate to reach a two-way conclusion with sufficiently high accuracy. In particular, we considered three-way decisions based on two probabilistic rough set models, namely, game-theoretic rough sets (GTRS) and information-theoretic rough sets (ITRS).

While this paper was, IMHO, not particularly strong, the idea is interesting, and we could do a similar thing (only better). In previous research we have developed many malware detection strategies, some of which are relatively costly. So, a good approach would be to use a "layered" defense, where a cheap score is used to filter out the easy cases, then progressively strong (but more costly) analysis is used on the harder cases. The project would involve testing this strategy against a purely binary classification approach. We might ultimately want to put this into the framework of "rough sets", but initially that would not be necessary.

75. Test high numbers of hidden states in HMM and compare to MCM (i.e., a straightforward Markov chain model). In a recent unpublished paper (A comparison of hidden Markov model and Markov chain model for malware classification), the authors claim the following.

> Recent research in the domain of malware analysis has seen growth in use of hidden Markov model (HMM) for tasks such as malware classification and clustering. Researchers have proposed various ways of exploiting the powerful pattern matching capabilities of HMM for differentiating malware from cleanware, as well as for classifying among malware families. In this research we extend the current state-of-the-art in HMM-based malware analysis in two dimensions. First we study the impact of number of hidden states on classification performance of a previously proposed HMM-based malware classification technique. Led by the observations from this experiment, we then propose to replace the HMM component of the classification method with a Markov chain model (MCM), and compare the proposed method with the HMM-based method from effectiveness and efficiency perspectives. Our comprehensive experiments on a dataset of real malware behavioral reports show that Markov chain model is a better choice than hidden Markov model for malware classification tasks. We also report that a combination of these two modeling methods can result in an improved performance as compared to the individual models.

Further, the authors claim that the best HMM has its number of hidden states equal to the number of distinct observation symbols (and they imply that the states correspond to the individual symbols, but don't verify it). And they claim that a similarly-configured MCM defeats the HMM more often than not. We could test all of these things using some of our many datasets.

76. A recent unpublished paper claims the following.

> Malware classification using machine learning algorithms is a difficult task, in part due to the absence of strong natural features in raw executable binary files. Byte $n$-grams previously have been used as features, but little work has been done to explain their performance or to understand what concepts are actually being learned.
>
> In contrast to other work using $n$-gram features, in this work we use orders of magnitude more data, and we perform feature selection during model building using Elastic-Net regularized Logistic Regression. We compute a regularization path and analyze novel multi-byte identifiers. Through this process, we discover significant previously unreported issues with byte $n$-gram features that cause their benefits and practicality to be overestimated. Three issues emerged from our work. First, we discovered a flaw in how previous corpora were created that leads to an over-estimation of classification accuracy. Second, we discovered that most of the information contained in $n$-grams stem from string features that could be obtained in simpler ways. Finally, we demonstrate that $n$-gram features promote overfitting, even with linear models and extreme regularization.

In effect, the paper shows that when a huge and diverse dataset is used, the features extracted from $n$-gram analysis are very generic (e.g., text strings that say "Microsoft"). This is not too surprising since generic data must yield generic models. In any case, it would be interesting to test $n$-gram analysis on progressively larger datasets (i.e., add more and more families) and see how the effectiveness (or lack thereof) behaves as a function of the amount (and diversity) of the data. Then it would be interesting to do the same thing, but with higher-level features such as opcodes.

77. A recent submitted paper (Testing Android Malware Detectors Against Code Obfuscation: A Systematization of Knowledge and Unified Methodology) looks at various obfuscations and analyzes their effectiveness against 6 different Android malware detection tools (Avast, Norton, Dr. Web, Kaspersky, Trend Micro, Zoner). One of the "obfuscations" considered is encryption. Assuming that only static techniques are used, encryption could be used to effectively remove a feature from consideration. A good project would consist of analyzing the effect of encryption of various features on a large Android malware dataset, over a wide variety of Android AV

products. This would enable us to, in effect, reverse engineer these AV products to determine what features they rely on for detection. With some additional work, we should be able to even estimate the amount of weight assigned to each of these features. The recommended dataset seems to be the Android Malware Genome Project (http://www.malgenomeproject.org), although they appear to no longer share their data.

78. A former student (Amrapali) developed a nested hill climb attack on a homophonic substitution. Someone (Markus) recently discovered a few bugs in the code and was also able to speed it up by a factor of 5. It would be interesting to compare this new and improved hill climb to an HMM with random restarts as methods for attacking homophonic substitution ciphers. We'd want to compare both success rates and work factors (based on timings). We would also want to test both programs on the unsolved Zodiac 340 cipher.

79. A recent (as yet unpublished) paper studies $n$-gram analysis of malware, where the $n$-grams consist of consecutive byte values. The authors use a vast dataset of malware and obtain poor results (in fact, the $n$-grams just seem to detect generic strings in Windows executables). As a result, they claim that $n$-grams are not a good feature to use for malware detection.

    I think there are a few problems with this paper. In particular, by using such a vast and generic dataset, they have turned the problem into that of detecting any generic malware, as opposed to detecting a specific malware family. This generic problem is sure to be hopeless, since at such a level, malware is likely to be so similar to general software as to be indistinguishable.

    I'd like to see the following experiments done. First, we would build models for each of several different malware families and test their effectiveness. Then, we would create models for combinations of families and test these (i.e., build models for all pairs of families, all sets of three families, and so on, up to a model that includes all families). We could build and test these models based on opcodes, and also do another set of experiments based on $n$-grams.

    It is likely that as we add more families, the models become more generic and hence less effective. It would be interesting to see if this is actually the case and, in any case, quantify the differences between the various models. We could compare the effectiveness of $n$-grams and opcodes, and the effectiveness of various machine learning algorithms.

80. A recent paper (not yet published) treats binary files as gray scale images and claims to generate a useful malware score from the resulting images. A good project would

consist of analyzing scores based on these images. Various image processing techniques could be tested to try to improve the score. The resulting scores would be compared to other scores, using challenging datasets.

81. A recent paper (unpublished as of now) uses a technique similar to our previous "stealthy ciphertext" (which can be found here: `http://www.cs.sjsu.edu/faculty/stamp/papers/stealthy.pdf`, and is used to encode encrypted email so that email filters will classify it as ordinary text-based email) to generate covert (and encrypted) network traffic, using DNS. Their technique may be an improvement, since it uses an HMM-based technique for conversion. A good project would consist of improving on the stealthy ciphertext result using the approach in this new paper.

82. A recent (unpublished) paper shows that network traffic generated by malware is highly periodic. While this may be interesting, the authors do not compare their results to traffic generated by benign applications. Consequently, we do not know whether periodicity of traffic is a distinguishing feature or not. A good project would consist of examining network traffic from a substantial number of malware applications and a large number of benign applications. Then, we can use a variety of machine learning techniques (and other approaches) to determine whether we can successfully distinguish malware applications from benign, based on periodicity (or lack thereof) of the network traffic that is generated.

83. It would be interesting to try to modify benign samples so that they appear to be malware, yet the programs function normally. Here's one possible approach: In several research papers we have shown that we can modify malware samples by adding dead code extracted from benign samples, thereby making the malware look more like benign code, and therefore harder to detect. Suppose that we instead extract code from malware samples and insert this into benign samples. This should have the effect of making the benign code look more like malware, causing some of the benign samples to be misclassified as malware. We could experiment to determine the most effective ways (from the attacker's perspective) to make such modifications. That is, we would like to minimize the modification, while maximizing the number of benign samples that are misclassified as malware.

84. The $K$-means algorithm gives us clusters that are circular in shape, while EM clustering using Gaussian (i.e., normal) distributions gives us oval shaped clusters. If we use EM clustering with the generalized hyperbolic distribution (GHD), we can obtain clusters where the ovals can be somewhat generalized to more of a diamond shape. There are even more general distributions that yield more general cluster shapes. These distributions are interesting and can be useful, but the number of parameters (based on the size of the covariance matrix) is large. So, convergence can be slow, and if we do not have sufficient data, the EM algorithm might not converge at all. To

reduce the number of parameters, and the amount of data needed for convergence, PCA can be used on the covariance matrix, independently for each cluster. Professor Cristina Tortora (cristina.tortora@sjsu.edu, who is in the math department at SJSU), has developed packages in R to generate clusters with these generalized distributions.

For this project, we would focus on clustering image spam and benign images. From a previous project, we have two image spam datasets, which we refer to as the standard dataset and the improved dataset, and we also have a benign image dataset. Each of these sets contains about 1000 images. There are some 21 features that have been extracted from each image. For the standard dataset, subsets of as few as 3 of these features have been shown to yield good detection results (based on SVM analysis). However, images in the improved dataset cannot be reliably distinguished from benign images.

The goal of this project is to determine how well we can cluster these image spam and benign datasets using the generalized distributions mentioned above (based on EM clustering). When clustering with these techniques, there are many parameters, and we would like to determine parameters that yield optimal results, with respect to the quality of the resulting clusters. Here, "quality" refers to the separation of image spam from benign—in the ideal case, all spam images are in a different cluster (or clusters) from all benign images. We would also want to experiment with the PCA approach mentioned above to reduce the number of parameters. It would make sense to experiment with all of the following cases.

- Standard image spam and benign datasets
- Improved image spam and benign datasets
- Standard and improved image spam datasets
- Standard and improved image spam datasets, and the benign datasets

In each of these cases, we would also want to experiment with various numbers of clusters. The standard/benign case should yield good results, and would serve as a control. If we can obtain reasonable results for the improved/benign case, this would show that we have sufficient information to distinguish the improved spam images from the benign images, something that we have not been able to do so far. The other cases would also be informative.

85. Here is the abstract from a paper that I recently reviewed. As of now, the paper is unpublished.

> Many efforts have been made to use various forms of domain knowledge in malware classification. Currently there exist two common approaches to malware classification without domain knowledge, namely byte $n$-grams

> and strings. In this work we explore the feasibility of applying neural networks to malware classification and feature learning. We do this by restricting ourselves to a minimal amount of domain knowledge in order to extract a portion of the PE header. By doing this we show that neural networks can learn from raw bytes without explicit feature construction, and perform even better than a domain knowledge approach that parses the PE header into explicit features.

The idea is that by parsing the PE header, we can obtain features for training, but this requires some "domain knowledge" (let's call this a "domain-full" approach). In contrast, we could simply use the raw bytes of the PE header (in the form of $n$-grams) to construct feature vectors (let's call this a "domain-free" approach). Specifically, the authors use tree-based machine learning techniques for their domain-full experiments, and neural networks for their domain-free experiments. They claim that the domain-free approach is better and yields better results. There are some problems with the paper however, one of which is that they are comparing a tree-based approach to a neural network technique, so it's an apples-to-oranges comparison. That is, it is just as plausible that the observed difference is due to the different machine learning techniques that are used, rather than the domain-free/domain-full dichotomy that they claim to be testing.

Another issue is that neural networks are extremely opaque, so it is difficult to gain any insight from the models themselves. Nevertheless, the authors argue (not very convincingly, IMHO) that their neural network models have learned essentially the same information as the tree-based models.

For this project, we would do various experiments with the goal of shedding some light on this topic. To begin with, we could train a neural network for each case (domain-free and domain-full), based on features extracted from PE headers, and compare detection results for these models. This would give us an apples-to-apples comparison of the domain-full case with the domain-free case.

We could then train a single model using both the domain-free and domain-full features (again, using a neural network), and see how its detection rate compares to both the domain-free and domain-full models. If we get significantly better results from the combined model, that would tend to indicate that there is some information available to each model that is not available to the other.

Additional experiments and various other machine learning techniques could be applied to this problem (HMM and SVM, for example). We could also consider other features, including opcodes versus raw byte $n$-grams within the `text` (i.e., `code`) section, as opposed to just focusing on the PE header.

86. Here is the abstract of a recent (possibly unpublished) paper that considers attacks on SVM-based malware classifiers.

    > Machine learning algorithms have been proven to be vulnerable to a special type of attack in which an active adversary manipulates the training data of the algorithm in order to reach some desired goal. Although this type of attack has been proven in previous work, it has not been examined in the context of a data stream, and no work has been done to study a targeted version of the attack. Furthermore, current literature does not provide any metrics that allow a system to detect these attack while they are happening. In this work, we examine the targeted version of this attack on a Support Vector Machine (SVM) that is learning from a data stream, and examine the impact that this attack has on current metrics that are used to evaluate a model's performance. We then propose a new metric for detecting these attacks, and compare its performance against current metrics.

    The idea is that the attacker manipulates the training data, with the goal is of causing the SVM to fail to detect a specific malware sample. In the SVM context, this has a nice geometric interpretation in terms of the separating hyperplane.

    A good project topic would consist of doing a "robustness" analysis of various machine learning algorithms (with respect to malware detection), where the attacker modifies the training data. It would be interesting to see how SVM, HMM, PCA, and various other machine learning algorithms compare in this respect.

    There are several ways in which such an attack could be conducted. For this project topic, we assume that an attacker can change the labels on the training data. That is, the attacker can insert malware samples into the benign dataset, and vice versa. For a related project, see topic number 87.

87. Project topic 86 proposes a robustness analysis of machine learning techniques (with respect to malware detection), where the attacker is able to change the classifications on training samples. In the malware context, another robustness issue arises when the attacker modifies the malware so that it is more like the benign set. For example, in several previous projects, we modify the malware samples to include dead code from benign samples, thus causing the statistical profile of the malware to more closely resemble that of the benign set. A good project would consist of carefully analyzing the robustness of various machine learning techniques (SVM, HMM, PCA, and so on) in this sense. That is, we would compare the strength of various machine learning techniques when the malware is made to look more similar to the benign set. This is somewhat similar to the work done in Tanuvir's project; see `http://scholarworks.sjsu.edu/etd_projects/409/` and `http://link.springer.com/article/10.1007/s11416-015-0252-0`.

This project would be most interesting if the same dataset and machine learning techniques are used as in topic 86. In that way, we could directly compare these two concepts of robustness on essentially the same datasets.

88. A recent paper compares data collected from Android malware via emulation versus similar data that is collected when the code is actually executed on an Android device. Here is the abstract.

    > The Android operating system has become the most popular operating system for smartphones and tablets leading to a rapid rise in malware. Many Android malware families employ detection avoidance techniques in order to hide their malicious activities from analysis tools. These include a wide range of anti-emulator techniques, where the malware programs attempt to hide their malicious activities by detecting the emulator. For this reason, countermeasures against anti-emulation are becoming increasingly important in Android malware detection. Analysis and detection based on real devices can alleviate the problems of anti-emulation. Hence, in this paper we present an investigation of machine learning based malware detection using dynamic analysis on real devices. A tool is implemented to automatically extract dynamic features from Android phones and through several experiments, a comparative analysis of emulator based vs. device based detection by means of several machine learning algorithms is undertaken. Our study shows that several features could be extracted more effectively from the on-device dynamic analysis compared to emulators. It was also found that approximately 24% more apps were successfully analyzed on the phone. Furthermore, all of the studied machine learning based detection performed better when applied to features extracted from the on-device dynamic analysis.

    A good project would consist of first verifying some of their results, that is, verify that executing the code on a real device does indeed give significantly more useful data than emulation. Assuming this is the case, then we would want to determine why this is so. The authors claim that it is due to anti-emulation techniques, but provide no evidence to support their claim (and I doubt that it is actually true, as most malware is not so sophisticated as to detect that it is being emulated).

89. Masquerade detection is a type of intrusion detection problem where we want to determine whether the current user is actually the currently logged in user. A recent paper (possibly unpublished) considers the problem of masquerade detection on a smartphone—although they use much different terminology. Here is the abstract.

> Improving the accuracy and reducing the time to authenticate users and preserving privacy are pivotal issues in smartphone security. A majority of owner identification methods have concentrated on improving accuracy and emphasize less on response time. Usage pattern of smartphone apps by the owner may be used as an important signature to differentiate between the legitimate and others. In this work, we rank the most informative apps specific to any owner to identify the owner using an information theoretic approach. Interestingly, with the reduced set of data related to highly ranked apps gives a higher detection accuracy with reduced learning time consumed by the classifiers.

This particular paper only considers very simple features, and only uses basic statistical analysis. It would be interesting to include more features and to use machine learning techniques (HMM and SVM, for example). I suspect that we could do much better than the results presented in this paper, and still have a very efficient and practical system.

90. Robust hashing for malware detection: The technique of "robust hashing" is used in image detection and is perhaps most famous for use in detecting child pornography online. The idea seems to be to hash small segments of files and use these to derive a sort of "signature" of the file, thus making the detection robust against common image distortion techniques. A good project would consist of using a similar approach for malware detection. One option would be to treat exes as images and then apply robust hashing directly to these images.

91. Implement and test USB attacks and/or defenses. Here is the abstract from a recent (as yet unpublished) survey of USB-based attacks.

> Attackers increasingly take advantage of innocent users who tend to use USB peripherals casually, assuming these peripherals are benign when in fact they may carry an embedded malicious payload that can be used to launch attacks. In recent years, USB peripherals have become an attractive tool for launching cyber-attacks. In this survey, we review 29 different USB-based attacks and utilize our new taxonomy to classify them into four major categories. These attacks target both individuals and organizations; utilize widely used USB peripherals, such as keyboards, mice, flash drives, smartphones etc. For each attack, we address the objective it achieves and identify the associated and vulnerable USB peripherals and hardware. In addition, we surveyed current detection and prevention solutions and found that they largely tend to concentrate on specific attacks or fail to provide a comprehensive and effective mitigation solution. Thus, in this paper, we also suggest a comprehensive, trusted, and multidisciplinary methodology

called USBEAT, which is aimed at the detection and prevention of known, and potentially unknown, USB-based attacks.

92. Followup to Vikash's paper on using image analysis (specifically, involving"gist" features and neural nets) for malware detection. We were unable to get good results using neural nets, but this should be possible with further experimentation.

93. Image spam analysis/detection based on "gist" features (see Vikash's paper for references on gist features and some basic background).

94. A recent (as yet unpublished) paper discusses the possibility of using an HMM based on a hierarchical Dirichlet process (HDP). This proposed research is described as follows.

    > The activity of a botnet runs through a set of phases: Formation, C&C, Attack, and Post-Attack. Depending on the botnet state, algorithms may vary the feature set used for detection and identication of botnets. Presumably the distribution of bytes and statistics associate with network flow features will change as a function of what state of activity the botnet is in. Thus a HMM would be ideal for capturing the different phases of botnet activity. The challenge for this research would be to capture data from botnets that encompasses the full range of their activity. Assuming it is possible to capture this data for one or more botnets, an HDP-HMM will be applied to these datasets to see if the data can be successfully clustered into botnet states.

    The mathematics behind HDP is fairly challenging (and obtaining useful data in the botnet case is another challenge), but it does seem plausible that this approach could be used to analyze botnets or other malware or security problems. For example, we might apply a similar analysis to code containing viruses to isolate the virus code from the benign code (in effect, separating the different "phases" of the code). Or we could apply this technique to masquerade detection (a type of intrusion detection problem), where we would want to distinguish the masquerader from the legitimate user.

95. A recent (as yet unpublished) paper discusses a method to detect malware that uses HTTPS. Since HTTPS is encrypted, few features are available for analysis. Here is the abstract of the paper.

    > One of the current challenges in network intrusion detection research is the malware communicating over HTTPS protocol. Usually the task is to detect infected end-nodes with this type of malware by monitoring network traffic. The challenge lies in a very limited number of weak features

> that can be extracted from the network traffic capture of encrypted HTTP communication. This paper suggests a novel fingerprinting method that addresses this problem by building a higher-level end-node representation on top of the weak features. Conducted large-scale experiments on real network data show superior performance of the proposed method over the state-of-the-art solution in terms of both a lower number of produced false alarms (precision) and a higher number of detected infections (recall).

The proposed technique only relies on four simple very features (uploaded bytes, downloaded bytes, duration, and inter-arrival time) and claims to achieve strong results.

In this previous work
`http://www.sciencedirect.com/science/article/pii/S0167404814000959`
we've developed a strong technique for detecting HTTP based attacks. Here is the abstract from that paper.

> Previous research has shown that byte-level analysis of network traffic can be useful for network intrusion detection and traffic analysis. Such an approach does not require any knowledge of applications running on web servers or any pre-processing of incoming data.
>
> In this paper, we apply three *n*-gram techniques to the problem of HTTP attack detection. The goal is to provide a first line of defense by filtering the vast majority of benign HTTP traffic, leaving only a relatively small amount of suspect traffic for more costly processing. We analyze these *n*-gram techniques in terms of accuracy and performance. Our results show that we can attain equal or better detection rates at considerably less cost, in comparison to a previously developed HMM-based technique. We also apply these techniques to a highly realistic dataset consisting of four recent attacks and show that we obtain equally strong results in this case. Overall, these results indicate that this type of byte-level analysis is highly effective and practical.

It would be interesting to compare these two approaches on a both HTTP and HTTPS traffic. We have the HTTP traffic, and we should be able to obtain the HTTPS traffic used in the research mentioned above.

96. Recently, there have been a few papers that apply image processing techniques to binaries (or various features extracted from binaries) as a means of obtaining high-level information that seems to be quite robust to a wide variety of obfuscations. That is, a binary executable (or some related feature or features) is treated as an image, and various image processing is done to the image (e.g., "gist descriptors").

Here is the abstract from one such paper (not yet published) that uses a very simple "image-based" representation derived from a control flow graph.

> To date, industrial anti-virus tools are mostly using signature-based methods to detect malware occurrences. However, advanced polymorphic viruses can change their signatures almost infinitely and effectively evade those tools by using packing techniques. In the research community, the approach of program analysis to detect suspicious behaviors has been emerging recently to handle this problem. Control flow graph (CFG) is a suitable representation to serve this purpose. However, the current typical CFG forms generated by state-of-the-art binary analysis tools, such as IDA, do not reflect the self-modification code methods used by most packing techniques. Moreover, this approach suffers from an extremely heavy cost to conduct and analyze the CFGs from binaries. This drawback causes the method of formal behavior analysis to be nearly not applicable with real-world applications.
>
> In this paper, we propose an enhanced form of CFG, known as lazy-binding CFG to dynamically reflect self-modification code behaviors. Then, with the recent advancement of the deep learning techniques, we present a method of producing image-based representation from the generated CFG. As deep learning is very popular to perform image classification on very large dataset, our proposed technique can be applied for malware detection on real-world computer programs and thus enjoying very high accuracy. We also illustrate our analysis results with some well-known malware samples, including EMDIVI and WannaCry.

There are many possible project topics using image techniques for malware detection. Another very interesting related topic would be to apply image-based techniques (together with machine learning) to classic ciphers. In particular, we would want to find a way to identify a classic cipher based on such features. For example, consider the digraph distribution matrix of a simple substitution cipher. This can be viewed as a graph, and we can derive an image based on this graph. From properties of this image, can we reliably determine whether the digraph distribution corresponds to a simple substitution or, say, random text, or text from another language? If so, can we extend this to homophonic substitution ciphers, and possibly other classic ciphers? If we can develop a reliable test for homophonic substitutions (as seems likely), then we can do some very interesting experiments on the unsolved Zodiac 340 cipher.

97. Here is the abstract from a recent paper on click fraud, titled "Click Fraud Detection on the Advertiser Side"

> Click fraud—malicious clicks at the expense of pay-per-click advertisers—is posing a serious threat to the Internet economy. Although click fraud has attracted much attention from the security community, as the direct victims of click fraud, advertisers still lack effective defense to detect click fraud independently. In this paper, we propose a novel approach for advertisers to detect click frauds and evaluate the return on investment (ROI) of their ad campaigns without the helps from ad networks or publishers. Our key idea is to proactively test if visiting clients are full-fledged modern browsers and passively scrutinize user engagement. In particular, we introduce a new functionality test and develop an extensive characterization of user engagement. Our detection can significantly raise the bar for committing click fraud and is transparent to users. Moreover, our approach requires little effort to be deployed at the advertiser side. To validate the effectiveness of our approach, we implement a prototype and deploy it on a large production website; and then we run 10-day ad campaigns for the website on a major ad network. The experimental results show that our proposed defense is effective in identifying both clickbots and human clickers, while incurring negligible overhead at both the server and client sides.

A possible project would consist of first breaking the detection technique used in this paper, then developing an improved technique. The proposed detection scheme is fairly ad hoc, and it should not be too difficult to defeat. And, I believe that by using various machine learning techniques, we can improve on these authors' results.

98. A recent (as yet unpublished) paper claims to use HMMs to "unmask" bitcoin transactions. Here is the abstract:

> With the rise of cryptographic ransomware, Bitcoin has found a niche as the standard currency for ransoms. While Bitcoin is pseudonymous, it provides no guarantee of untraceability. As a result, another niche has arisen—Bitcoin money laundering. Hidden Markov Models (HMMs) have previously been used in a number of applications where traditional pattern recognition falls short. In this paper, HMMs are inferred from transactions in the public blockchain in an attempt to link users, events, and enterprises. We introduce a proof-of-concept algorithm to infer HMMs from the Bitcoin blockchain.

An interesting project would consist of using HMMs to try to unmask bitcoin money launderers, with lots of experimental results to demonstrate the feasibility and/or difficulties with such an approach.

99. Attempt to break a combination cipher consisting of a homophonic substitution and a column transposition (HSCT) as follows. Find the best transposition for the HSCT using one or both of the scores discussed here (under number 4, "breaking transposition ciphers"):

    `http://echidna.maths.usyd.edu.au/kohel/tch/MATH3024/Tutorials/Solutions/tutorial05.pdf`

    These scores should still be strong, even if the plaintext is encrypted with a simple substitution, which we can easily test. For a homophonic substitution, the scores will be much weaker, but they should still reveal some useful info. Again, this needs to be tested. Assuming this works, for the highest scoring column transpositions, we can then attack resulting homophonic substitution cipher using an HMM with multiple random restarts. These experiments are aimed at an attack on the Zodiac 340 cipher. For testing purposes, we can permute the columns of the Zodiac 408 cipher.

100. Apply alternating least squares (ALS) to the malware detection (or classification) problem (details TBD).

101. Study the use of HMMs (with multiple random restarts) for cryptanalysis of polyalphabetic substitution ciphers (and possibly also ciphers that include a combination of substitution and elementary transpositions). The Vigenere cipher is a good place to start.

102. A recently submitted paper discusses using accelerometer data from a smartphone as a form biometric authentication. The authors suggest having the user make various gestures (i.e., trace out letters or symbols) with the phone, and analyze the accelerometer data. Here is the (poorly written) abstract from this paper:

    > Recently, mobile devices such as smart phones have been spreading widely. Privacy sensitive data is stored on those devices. Computer security becomes a serious problem on mobile devices. Thus biometric-based methods for mobile devices are required in order to prevent the leakage of important data. In this paper, we proposed gesture-based authentication system. We focused on the characteristics of acceleration when the user drew letters or symbols in the air. We implemented and evaluated a biometric-based system using this characteristics. Dynamic Time Warping using Euclidean distance is often used when we calculate the similarity. However, this method has the problem that the Euclidean distance is unintentionally similar even if the user performed different gestures. In order to deal with this problem, we proposed a similarity calculation method by Dynamic Time Warping using cosine similarity. We evaluate the effectiveness of our method by comparing the performance of these methods.

The authors fail to consider realistic attacks (e.g., an attacker who sees a user's authentication gestures and tries to mimic those gestures). A good project would consist of trying to break this authentication scheme (which I suspect will not be too difficult). Then we would try to improve on their authentication scheme by using machine learning techniques to strengthen the scoring.

103. We recently submitted a paper on image spam analysis, that is based on SVMs. In that paper, we discuss a "challenge" dataset that we constructed, which contains image spam that is designed to be difficult to detect. A good project would consist of testing a wide variety of machine learning techniques on this dataset, and trying to develop new (and stronger) features that will help in detecting difficult image spam, such as that found in this challenge dataset. Most likely, edge-based features will be most useful for this problem.

104. This article discusses "Blockchain-based Machine Learning Marketplaces":

     `https://medium.com/@FEhrsam/blockchain-based-machine-learning-marketplaces-cb2d4da`

     The idea is to combine blockchain technology with "secure computation.." Then machine learning can be applied to private data (and it will remain private, thanks to "secure computing") and people can get paid for their private data, thanks to the use of blockchain techniques. The ideas are pretty vague, but the bottleneck is in the "secure computing" aspects (the author lists 3 techniques, but they are all slow and/or not clearly applicable to the problem). At this point, I don't have any clear idea for a well-defined project in this area, but if you are interested, we can dig deeper to see what might be do-able within the limitations of a master's project.

105. Researchers often combine various classifiers in a variety of ways. This classic paper discusses some types of combinations of classifiers and comes up with a framework for various (more or less) ad hoc combinations:

     `https://dspace.cvut.cz/bitstream/handle/10467/9443/1998-On-combining-classifiers.pdf`

     That's a 20 year old paper, and when I look it up on Google scholar and click the "cited by" link, I don't see a lot of relevant modern research on this topic (of course, I might have missed something). In any case, it would be interesting to look at this problem from a more modern perspective. It would be nice to have a theoretical framework that would include SVM, boosting, and random restarts (with an HMM, for example), along with the (relatively) ad hoc methods that are considered in the classic paper cited above. I have written a paper that cans serve as a framework for this part of the project. For a master's project, you could do empirical studies, comparing results obtained using various "ensemble" techniques applied to the malware detection problem (or some other problem domain). For example, we would like to do a direct comparison of SVM, boosting, random restarts, and each of the ad hoc techniques in the paper cited above on a large malware dataset that we have available. Another option would be to do these experiments using a large image spam dataset, or for the masquerade detection problem, for example. Many other such experiments involving various problem domains and a variety of machine learning techniques would fit into this general research area.

106. For some problems, we have more than one hill climb technique available. In such cases, it would make sense to "ping pong" back and forth between these two hill climb techniques (call it a PPHC algorithm, for "ping pong hill climb"). That is, we run one hill climb until it can climb no further, then we switch to the other hill climb until it climbs no more, then switch back to the first hill climb, and so on, until neither hill climb can improve on the solution. Note that this is somewhat analogous to the process used in $K$-means clustering, where we switch back and forth between two different techniques, each of which can be viewed as hill climbing (i.e., the solution can only improve—it can never get worse).

    As a test case for the PPHC strategy, we could first experiment with a combination of an HMM and Jakobsen's algorithm, applied to simple substitution cryptanalysis. Previous work has been done to carefully analyze the success rate of Jakobsen's algorithm and HMM (with random restarts) when applied to simple substitution cryptanalysis; see `https://www.tandfonline.com/doi/abs/10.1080/01611194.2015.1126660`.

    Both HMMs and a simple substitution distance (SSD) score based on Jakobsen's algorithm have been successfully applied to the malware detection problem; see `https://link.springer.com/article/10.1007/s11416-013-0184-5` for details the SSD score. We could experiment with the PPHC techniques for malware detection, and compare the results to an HMM score and the SSD score, as well as other machine learning based scores.

    Also, a PPHC approach could be applied to homophonic substitutions, since there is a generalization of Jakobsen's algorithm for such ciphers (see `https://www.tandfonline.com/doi/abs/10.1080/01611194.2013.797041`) and HMMs (with random restarts) can also perform well against homophonic substitutions. In this case, it might be most effective to use an HMM (with fixed $A$ matrix of size $26 \times 26$) to simply determine the distribution of plaintext letters to ciphertext symbols (i.e., how many ciphertext symbols map to plaintext `E`, and so on), as this is the most challenging part of the generalized Jakobsen's algorithm for homophonic substitutions. We could also experiment with a more basic version of an HMM with $N = 2$, just to obtain the distribution between consonants and vowels, and use this information in the homophonic substitution version of Jakobsen's algorithm. We would want to test all of these ideas on the solved Zodiac 408 cipher, and on the unsolved Zodiac 340 cipher.

107. In machine learning, the "cold start" problem deals with training models when limited data is available. For example, in the malware context, we might initially only have a very limited number of samples from a particular family, but we'd like to train a model on this limited dataset for detection, instead of waiting (without any protection) until we have a large training set. It would be interesting to experiment with different techniques (HMM, SVM, RF, k-NN, boosting, etc.) to see which are most effective for this cold start malware problem.

    There are many possible extensions of this topic. For example, in previous work on HTTP attack detection, we found that PHMMs were most effective in the cold start situation, but once sufficient data was available, an HMM would outperform the PHMM, and we determined the crossover point, where HMMs become the better choice. This would be interesting with respect to the malware problem and with respect to other ML techniques.

    As another extension, we could try to merge various cold start models to see how well they perform on more generic datasets. In general, we expect that as the data becomes more generic (i.e., as more models are merged), the detection problem becomes more difficult, and we have studied this for malware, but not in the cold start (i.e., limited training data) case. Extending this to the cold start case, and we might call it the "cold fusion" problem, as we are merging (fusing) multiple families into one model.

108. Suppose that we have 3 malware families, call them family A, family B, and family C. If we want to detect all of these families, we could train a model for each (model A, model B, and model C) and then test against all of three models, and classify as malware or benign based on the results (using a simple voting and/or thresholding scheme, or a "dueling HMM" approach, or a "multi-sensor" approach, or ...). On the other hand, we could train an entirely new single model on families A, B, and C and use the resulting model. We could refer to using multiple models as "cold fusion", as we are combining existing models without retraining, and we can refer to training a new model for all three families as "hot fusion", since we need to train a new model. A good project would consist of comparing these hot and cold fusion strategies (both in terms of accuracy and efficiency) on, say, a large and diverse malware dataset, and over a variety of machine learning techniques.

109. Here is an idea for a project related to malware evolution—see topic number 48 for a different type of malware evolution project idea.

    Here is the abstract from a paper that I recently reviewed.

> Android malware is increasing in spread and complexity. Advanced obfuscation, emulation detection, delayed payload or dynamic code loading are some of the techniques increasingly employed by current malware to hinder the use of reverse engineering and anti-malware tools. This growing complexity is particularly noticeable in the evolution of different strands belonging to the same malware families. Over the years, these families mature to become more effective, successively incorporating new and enhanced techniques. In this paper we focus on a particular family of Android ransomware named Jisut and perform a thorough technical analysis, providing also a detailed overall perspective which will help to create new instruments to tackle the threat posed by this ransomware more effectively.

In this paper, the authors claim (not very convincingly, IMHO) that they can trace changes in different variants of a specific ransomware family by reverse engineering code samples and examining them carefully. I'd like to examine code samples and look for such evolutionary changes, but using automated and quantifiable techniques based on machine learning. Here is one such idea.

One student recently collected a large and diverse malware dataset. We'll select a malware family from this dataset and tag each sample in this family according to the date on which it was collected. Then we'll train a linear SVM based on byte $n$-grams (possibly other features) over all samples from, say, a 1 year time interval. Using this model, we can then rank (or quantify the importance of) all features based on the SVM weights. Next, we shift ahead 1 month and again train an SVM on all samples within a 1-year window, and again rank the features. Continuing, we obtain a series of snapshots (based on overlapping sliding windows, each of 1 year duration, and each offset by 1 month) consisting of ranked features. We can then trace changes in the significance of various features and use this as a means to quantify changes in the malware family—we should be able to detect both slow evolutionary changes, and sudden breaks in the evolution as well. By using $n$-grams (or other intuitive features), we can relate the detected evolutionary changes in the models back to the code itself.

In this project we would want to experiment with a wide variety of malware families, and with a variety of features. It would also be interesting to experiment with several different machine learning techniques.

110. Train HMMs and apply $k$-NN to the resulting models. For example, we could train an HMM for each of several malware families in our set of training data. Then when a new (or unknown) family appears, we could train a model on this new family, and classify it based an $k$-NN. That is, we would use $k$-NN to determine its nearest neighbor(s) in the training set models, and classify it accordingly. In this way, we

might be able to distinguish new families that are variants of known families. A similar approach could be used based on $K$-means or other clustering algorithms. In either case, it would be important to use a metric that accounts for the structure or models generated by HMMs, where the order of the hidden states can vary (e.g., hidden state 1 of a model could be hidden state 2 in another model for the same family, based on different initial parameters or data). Note that Rabiner's classic HMM tutorial (`https://www.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/tutorial%20on%20hmm%20and%20applications.pdf`) includes a discussion of ways to compare HMMs (Section IV, part F).

111. A student recently completed a project on robust hashing (see topic 90) based on treating malware executables as images. There were quite a few loose ends, and lots of ways that we could go with additional research projects on this topic. For example, we could look at uncompressed image formats, with should be more informative. Also, we could apply clustering techniques for the "compression" step rather than the coding (and wavelet) approaches that were used in the previous project. More generally, we'd like to better integrate machine learning into this robust hashing strategy.

112. This graph purports to show the relationship between deep learning and other machine learning techniques as a function of the amount of training data available: `https://ibb.co/m2bxcc` (this graph is from the deep learning tutorial at Kaggle, `https://www.kaggle.com/kanncaa1/deep-learning-tutorial-for-beginners`). Since we now have a large malware dataset, we can test this empirically for the malware problem. A good project would consist of training various machine learning techniques (SVM, HMM, deep learning) and comparing the resulting accuracy (and AUC) as a function of the amount of data used for training. We could also consider analogous robustness experiments, where we manipulate the training data to make the detection problem more challenging. In any case, we would like to know where the crossover point lies (i.e., how much data do we need before deep learning becomes more effective than non-deep learning), and we'd want to know how large the advantage for deep learning can grow as we add more and more training data.

113. Some recent work claims that they can distinguish malware from benign over a large number of malware families based on $n$-gram features. It would be interesting to try something similar using deep learning and image-based analysis, as a followup to Sravani and Vikash's paper, for example.

114. In neural networks there is a concept known as transfer learning, where a model is trained for one problem, then retrained (at a much lower cost) for a somewhat different problem. Recently one of my students used this technique to detect malware samples that were considered as image files. The original models were trained on image

datasets consisting of vast numbers of diverse images, which was a very slow and costly process. Retraining these models for malware "images" was then relatively fast and produced excellent results; those results can be found here `http://www.scitepress.org/Papers/2018/66858/66858.pdf`.

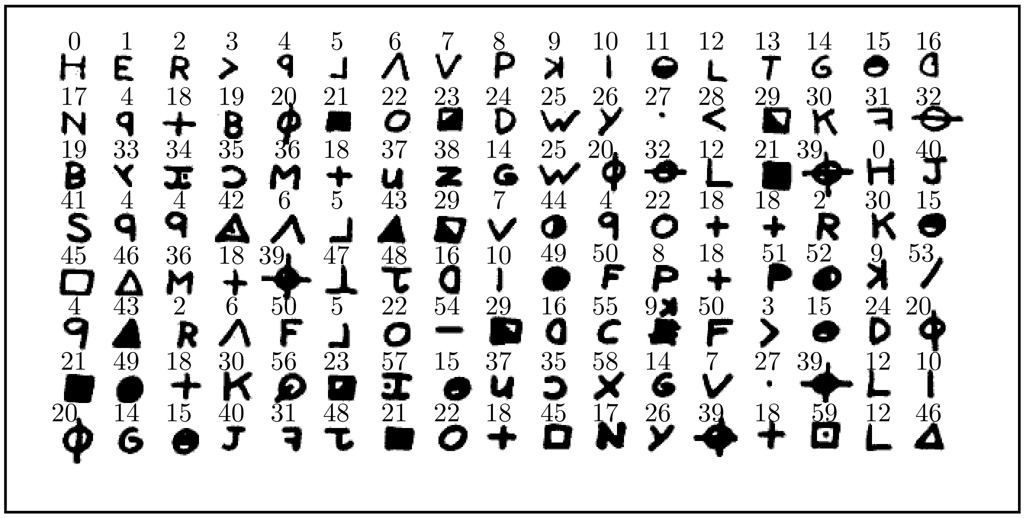
There are many problems where it is advantageous to train large numbers of HMMs using different initial values (typically, with random restarts). It is possible to view the HMM training process in terms of Lagrange multipliers, and such a problem can be solved using neural networks—see Section 7.4.2 of Rojas' book for a brief discussion `https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf` and Section 6.1 of my deep learning tutorial `https://www.cs.sjsu.edu/~stamp/ML/files/ann.pdf` for more details. This paper might provide an alternative (and perhaps better) approach to viewing HMM training as a neural network problem `https://core.ac.uk/download/pdf/4881023.pdf`; see the last homework problem in my tutorial `https://www.cs.sjsu.edu/~stamp/ML/files/ann.pdf` for more details on this algorithm.

If we can apply transfer learning to the HMM training problem, we might be able to train large numbers of HMMs very efficiently. For this project, you would want to show that transfer learning is applicable to the HMM training problem, and then test it in a specific problem domain (such as malware detection or solving certain types of ciphers, among many others), where HMMs with random restarts can yield significant improvements.

115. This paper `https://core.ac.uk/download/pdf/4881023.pdf` claims to provide an method for training HMMs that can be used in an "online" mode; see the last homework problem in my tutorial `https://www.cs.sjsu.edu/~stamp/ML/files/ann.pdf` for more details on this algorithm. For an online HMM training algorithm, we would be able to update the model as more training data becomes available, without having to retrain from scratch. This could be advantageous in a lot of settings. For example, if we want to use HMMs for intrusion detection, we could train a model as soon as the available data reaches a predetermined threshold, then update the model as more data becomes available. For this project, you would first need to implement and test the online version of HMM training, and then we would apply it to one or more specific problems, such as masquerade detection or malware detection.

116. A person who has done a lot of work on classic ciphers recently claimed to have solved the Zodiac 340 (Z340) cipher; see `https://www.youtube.com/watch?time_continue=1115&v=2WFLrQTrOtO` for more details. However, his putative solution seems to be based on some highly questionable assumptions. This putative solution is based on the (reasonable) assumption that most of the ciphertext is filler, with only the first 8 lines (and part of one final line) out of the 20 lines being decipherable.

The first 8 lines of the Z340 cipher appears below, where the numbers correspond to the ciphertext symbols.



The putative solution for the first 8 lines is given below, where the numbers above are (as above) my interpretation of the ciphertext symbols.

| Row | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0<br>H | 1<br>E | 2<br>R | 3<br>E | 4<br>I | 5<br>T | 6<br>I | 7<br>S | 8<br>I | 9<br>K | 10<br>I | 11<br>L | 12<br>L | 13<br>B | 14<br>O | 15<br>T | 16<br>H |
| 2 | 17<br>N | 4*<br>I | 18<br>G | 19<br>H | 20<br>T | 21<br>A | 22<br>N | 23<br>D | 24<br>D | 25<br>A | 26<br>Y | 27<br>. | 28<br>I | 29<br>L | 30<br>I | 31<br>V | 32<br>E |
| 3 | $\boxed{19}$<br>B | 33<br>Y | 34<br>T | 35<br>H | 36<br>E | 18*<br>G | 37<br>U | 38<br>N | $\boxed{14}$<br>B | 25*<br>A | $\boxed{20}$<br>R(R)E | 32* | 12*<br>L | 21*<br>A | 39<br>I | $\boxed{0}$<br>M | 40<br>␣ |
| 4 | 41<br>S | $\boxed{4}$<br>O | $\boxed{4}$<br>Q | 42<br>U | 6*<br>I | 5*<br>T | 43<br>W | $\boxed{29}$<br>I | 7*<br>S | 44<br>H | 4*<br>I | 22*<br>N | 18*<br>G | $\boxed{18}$<br>F | $\boxed{2}$<br>O | $\boxed{30}$<br>R | $\boxed{15}$<br>G |
| 5 | 45<br>A | 46<br>M | 36*<br>E | $\boxed{18}$<br>T | $\boxed{39}$<br>O | 47<br>B | 48<br>E | $\boxed{16}$<br>O | $\boxed{10}$<br>V | 49<br>E | 50<br>R(P)I | 8* | 18*<br>G | 51<br>I | 52<br>S | $\boxed{9}$<br>␣ | 53<br>M |
| 6 | 4*<br>$\boxed{I}$ | 43*<br>W | 2*<br>R | 6*<br>I | $\boxed{50}$<br>S | 5*<br>T | 22*<br>$\boxed{N}$ | 54<br>$\boxed{I}$ | 29*<br>L | 16*<br>O | 55<br>C | 9*<br>K | 50*<br>S | $\boxed{3}$<br>? | $\boxed{15}$<br>N | $\boxed{24}$<br>O | $\boxed{20}$<br>W |
| 7 | 21*<br>A | $\boxed{49}$<br>N | 18*<br>G | 30*<br>R | 56<br>Y | 23*<br>D | 57<br>A | 15*<br>N | $\boxed{37}$<br>G | $\boxed{35}$<br>E | 58<br>R | 14*<br>O(U)S | 7* | 27*<br>. | $\boxed{39}$<br>␣ | $\boxed{12}$<br>␣ | 10*<br>I |
| 8 | 20*<br>W | 14*<br>O | 15*<br>N | $\boxed{40}$<br>T | $\boxed{31}$<br>C | $\boxed{48}$<br>H | 21*<br>A | 22*<br>N | 18*<br>G(E)A | 45* | 17*<br>N | 26*<br>Y | 39*<br>O | 18*<br>F | 59<br>G | $\boxed{12}$<br>A | 46*<br>M(E) |

A good project would consist of analyzing this putative solution to the Z340. To start with, we could train an HMM based on the observation sequence above (i.e., the numbers above the putative plaintext symbols). That is, we could train large numbers of HMMs with random restarts to determine the best solution that we can find by using HMMs. It would be interesting to compare the best solution that we can find to the putative solution given above. It would be even more interesting if we can allow for "reverse homophones" in the HMM, that is, if we allow some number of symbols to decrypt inconsistently (all of the boxed numbers in the putative decrypt above are reverse homophones). This would make it much easier to find a solution, provided that we can incorporate this idea into the HMM training.

117. A recent (and excellent) paper titled "Anomaly detection techniques based on kappa-pruned ensembles" discusses some advanced methods for combining classifiers into an ensemble. They apply their technique to the problem of intrusion detection. Interestingly, their research is based on HMMs, and the authors claim that by combining HMMs trained with different numbers of hidden states into an ensemble, significantly better results can be obtained, as compared to the individual classifiers. This is somewhat reminiscent of what we did in a recent student paper (`https://link.springer.com/article/10.1007/s11416-018-0322-1`), where we compared AdaBoost, based on HMMs trained with random restarts, to the case where we simply use the best HMM from among the random restarts. However, we used the same number of hidden states for all of our models, which in retrospect does not look like the best approach.

    For this project, we can attack the problem from a couple of different angles. First, we would again generate a lot of HMMs using random restarts, but we would also vary the number of hidden states. Then we would want to compare the case where we select the best HMM (or a mini-ensemble based on the "best" model for each value of $N$, where $N$ is the number of hidden states) to a big ensemble. For generating the ensemble (or ensembles) there are a lot of possibilities, including some of the sophisticated things done in the "kappa-pruned" paper cited above. But, I am not convinced that there is great value in complex ensemble techniques, as compared to something more straightforward, such as an SVM, or some other ML technique, perhaps ANNs. That is, we would also want to experiment with a variety of ensembling (if that's a word) techniques, from simple to relatively complex. I suspect that by using ML techniques to generate the ensemble itself, we can obtain results comparable to—if not better than—using complex ensemble techniques. It would be interesting to see if this is really the case.

    For these experiments, we could work with any data, but malware would be a good choice, since we have a lot of experience in this domain, and we have a new and extremely large dataset.

118. Word2Vec is a technique (based on a shallow neural network) for embedding words into a high dimensional space (typically, 100s to 1000s of dimensions). The idea is that the closer that the words are in this new space, the more closely related are the words. There has been some effort to apply Word2Vec to other domains, but the

only malware-related application that I've seen is `https://ieeexplore.ieee.org/document/8071952`, which treats "machine code instructions" as words.

A good project would consist of mapping malware samples using a technique modeled on Word2Vec. We would want to experiment with different features as "words" (opcodes and $n$-grams would be easy, but not very informative, while treating the entire malware sample as a "word" would be ideal, but might be computationally challenging). Given such a mapping, we could then experiment with the resulting embedding technique to see how well it classifies malware samples into their respective families. In this case, we would project samples into the Word2Vec space and classify based on the nearest neighbor(s) in a given training set. This could also be viewed as a form of clustering. In addition, we would experiment with (simulated) zero day malware, where we attempt to classify malware that is not in the training set. Techniques other than $k$-nearest neighbor could be used for classifying (clustering). Finally, we would want to compare this Word2Vec technique to SVM, which involves a somewhat similar strategy of mapping to a higher dimensional space.

119. Suppose that we are given ciphertext that we suspect was generated by a classic cipher, but we don't know which one. Can we train a model so that it can identify the cipher type with high accuracy? Various models could be tested, including a multiclass SVM, but probably the most interesting case would be a neural network.

Here is one possible approach to using a neural network to solve this problem.

- First, train a large number of HMMs on different simple substitution ciphertexts (i.e., different keys), and also train a large number of models on different Vigenere ciphertexts. For each HMM, use $N = 2$ hidden states, $M = 26$, and a large value of $T$ (like 50k, to be sure the models converge), and enough iterations so that the models converge (typically, 200 is enough, but you might use 500, assuming your HMM code is fast).
- Next, we want to treat each HMM (just the $A$ and $B$ matrices, as the $\pi$ matrix probably won't provide any useful info) as a greyscale image, where each "pixel" is shaded, based on the value in that position of the matrix.
- Finally, we want to train a convolutional neural network (CNN) to see how well it can distinguish between the two classes (simple substitution vs Vigenere), based on the HMM images from the previous step.

The idea here is that CNNs are very good at detecting patterns in images, regardless of where the pattern appear in the image (e.g., an image will be recognized as containing a picture of a cat, regardless of where the cat actually appears within the image). By taking advantage of this property, we might be able to recognize similar structure within each class that would otherwise be difficult to detect (due to inconsistencies in the ordering of the hidden states, different keys having been used, and so on).

I suspect that this technique will work better when $N$ is larger, since we have more detailed "images" in such cases. So, we would want to repeat these experiments for larger $N$, up to (and including) $N = 26$.

Of course, if this works to distinguish between simple substitution and Vigenere ciphertexts, we would want to extend it to include other classic substitution and transposition ciphers. Examples of classic substitution ciphers include the simple substitution (which includes the Caesar's cipher and ROT13 as special cases), Vigenere cipher, affine cipher, Hill cipher, and Playfair cipher. There are also many classic transposition ciphers, including the columnar transposition, double transposition, rail fence cipher, and route cipher.

120. A variant of an HMM known as a GMM-HMM (Gaussian mixture model-hidden Markov model), or sometimes given as HMM-GMM, is popular in speech research. In a GMM-HMM, we replace the $B$ matrix with a collection of Gaussian (i.e., normal) distributions. In this formulation, there are $N$ Gaussian distributions (one for each hidden state), and each distribution is multivariate, with $M$-dimensions, where $M$ is the number of distinct observation symbols. To train such a model, we can determine the $A$ and $\pi$ matrices using the well-known Baum-Welch re-estimation process (or using a lesser-known gradient ascent technique). But, instead of re-estimating the $B$ matrix, we need to estimate the parameters (specifically, vectors containing the means and variances) for each of the unknown Gaussian distributions.

While it might seem difficult to determine the parameters of the Gaussian distributions, it is actually the same process as is used in EM clustering, which is described in detail in Section 6.5 of my machine learning textbook, *Introduction to Machine Learning with Applications in Information Security.*

As mentioned above, the GMM-HMM approach is widely used in speech and audio problems, but it does not seem to have been applied in many other fields, For example, in the malware research domain, GMM-HMMs do not seem to have been studied. It would be interesting to compare the performance of standard HMMs to GMM-HMMs for malware detection and classification problems. There are many good malware datasets and a variety of features where HMMs have been considered, so we have a good idea as to how they perform. We would like to determine whether the additional complexity of a GMM-HMM adds any significant value, as compared to a standard HMM, and we would also want to compare to other ML techniques, such as SVM and, possibly, DNN.

121. In a monoalphabetic substitution cipher, the encryption key does not vary throughout the encryption process. The simplest example of such a cipher is a simple substitution,

with a homophonic substitution being a slightly more complex example. In contrast, for a polyalphabetic substitution, the substitution changes. A simple example of a polyalphabetic substitution is the Vigenere cipher, and WWII-era cipher machines (such as the German Enigma) are more sophisticated examples of polyalphabetic substitutions.

In a previous paper, we showed that an HMM is an effective tool to break a Vigenere cipher (`http://www.ep.liu.se/ecp/149/011/ecp18149011.pdf`). A recent paper applies a genetic algorithm (and other heuristic search algorithms) to the Vigenere cryptanalysis problem (A Learned Polyalphabetic Decryption Cipher, by C. Hewage, et al.). A good project would consist of first comparing the effectiveness of an HMM to a genetic algorithm when applied to the problem of breaking a Vigenere cipher. In particular, we would be interested to determine which performs better when the available ciphertext is limited. Then we would want to extend this analysis to more advanced polyalphabetic substitutions.

122. Generally, training machine learning models is costly and slow, while the scoring (or classification) phase is fast and efficient. The goal of real-time machine learning (RTML) is to be able to also train models in real time. This might be useful in applications where the model needs to be frequently updated, based on recent data. In this project, we consider a semi-RTML approach that is modeled on the way that long-term and short-term memories are thought to be stored by the brain.

A deep neural network-hidden Markov model (DNN-HMM) is a combined technique, where the output of a trained DNN is used to create a time series that is then used as the input to an HMM. Training a DNN is costly, and in most realistic scenarios is not something we can do in real-time. Also, the usual method of training an HMM (i.e., Baum-Welch re-estimation) would be challenging to do in real time, since the model must be retrained from scratch each time we want to update the model. However, there is another way to train an HMM based on a gradient ascent that can be used in an "online" mode, meaning that the model can be updated as new training data becomes available, without the need to retrain on the previous data (for example, see this tutorial: `https://www.cs.sjsu.edu/~stamp/RUA/hmmOnline.pdf`). This gradient ascent/online version of training an HMM could be used in a real-time situation, particularly when an existing model simply needs to be updated to reflect recent observations.

For this project, we would want to implement a DNN-HMM then update the HMM part of the model in real time as new training data becomes available. The DNN part of the model could possibly be updated if there are extended time periods when spare CPU cycles are likely to be available, but this will depend heavily on the specific application being considered.

There are many possible problem domains where this technique could be considered. Since we have many good malware datasets, perhaps we could experiment with recent Android malware dataset. Although real-time updates for android malware detection might not be critical, the fact that you can easily and efficiently update the model (at least the HMM part) could be relevant, assuming the detection is taking place on the phone. In general, this DNN-HMM approach would allow for the use of a powerful and easily updated model in a resource-constrained environment. There are many other possible application domains that could be considered.

The previously mentioned connection to humans memory is as follows. Humans have both short-term and long-term memory. It is thought that sleep plays an important role in the formation of long-term memories. In our DNN-HMM model, we could view the HMM part as playing the role of short-term memory. The DNN part of the model is like long-term memory, which only gets updated during down times. Memory is not exactly the same as learning, but it is close enough for this analogy.

123. A paper that I recently reviewed considered the problem of image-based malware analysis. We have done similar projects; see, for example, topics 80 and 92, above, and these two papers (co-authored by previous students):

    - N. Bhodia, et al, Transfer learning for image-based malware classification, 3rd International Workshop on Formal Methods for Security Engineering (ForSE 2019), in conjunction with the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019), Prague, Czech Republic, February 23–25, 2019.
    - S. Yajamanam, et al, Deep learning versus gist descriptors for image-based malware classification, 2nd International Workshop on Formal Methods for Security Engineering (ForSE 2018), in conjunction with the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018), Funchal, Madeira, Portugal, January 22-24, 2018.

    The paper that I reviewed was not impressive overall, but it did have one interesting idea. Whereas previous work uses greyscale images, this paper claimed to use color images (although it was not clear how the color images were generated, or that their approach provided any benefit over greyscale).

    Since each pixel in a color image consists of three bytes (i.e., RGB colors), we could generate color images based on overlapping windows of three consecutive bytes from `exe` files. Then, we could apply convolutional neural network (CNN) techniques to such images obtained from malware samples. This would provide an interesting CNN model based on 3-grams. While both CNNs and n-grams have been studied in the context of malware detection, this combination does not seem to have been previously considered.

124. A recent paper discusses a method for combining the results of mulitple AV scanners to generate a superior scanner, based on a process that they refer to as a greedy approximation model (GAM), which is based on combined probability model (CPM). The authors claim that their GAM technique can generate near-optimal results (by some measure) with reasonable efficiency.

    This CPM technique assumes that the AV scanners are independent. Given any reasonable number of scanners, it is virtually certain that the scanners are highly *dependent*, as virus scanners tend to rely on similar features and are likely to also use similar statistical techniques to classify samples. The GAM technique gets around this independence restriction to some extent, but requires that all pairwise dependencies are known. In contrast to this GAM approach, machine learning techniques such as boosting, for example, should work well to generate near-optimal combined AV scanner results—regardless of whether the component scanners are independent or highly dependent, or whether we know anything about these dependencies a priori.

    For this project, you would create AV scanners that consist of combinations of AV scanners from, say, VirusTotal, using a GAM (or similar) statistical technique. Then you would also create such scanners based on the same set of AV scanners using various machine learning techniques (boosting, and possibly SVM, $k$-NN, random forest, neural networks, etc.). You would compare the results of these two approaches (primarily in terms of accuracy, but we would also want to consider efficiency and other metrics) based on a large and diverse malware dataset.

125. A recent paper discusses a "slippery" hill climb attack on periodic polyalphabetic substitutions, such as the Vigenere cipher. The attack is based on letter 4-grams using a variant of Jakobsen's algorithm, and the attack considers one permuation/alphabet at a time. A key idea is that the attack is not a strict hill climb, but instead at some steps the result can possibly un-improve. This approach can allow for a better solution than could be obtained using a strict hill climb, as it is possible to escape from a local minimum.

    It would be interesting to develop and analyze a "slippery HMM" or SHMM. We could add another parameter to an HMM that represents the probability of a "slippery" step. At each slippery step, we would do something to nudge the state off of its current trajectory. This could be something as simple as randomly changing a few of the freshly-computed elements of the $A$ and $B$ matrices (taking care to be sure that the matrices remain row-stochastic), or something more drastic, like swapping elements of the matrices.

Any proposed SHMM would need to be thoroughly tested on some specific problem domain. For example, we could consider classic ciphers, and perhaps even the same periodic polyalphabetic substitutions as used in the paper mentioned above. We could compare HMM with random restarts (HMMRR) to the SHMM to a combination of HMMRR and SHMM, with the ultimate goal being to find the optimal combination of the two. Another alternative would be to conduct analogous experiments on malware.

126. A recent submitted paper considers an Extreme Learning Machine (ELM) architecture for malware detection/classification. ELMs are a class of neural networks that essentially use random layers, this making the training fast (since no backpropagation is required) while sacrificing accuracy and stability. ELMs seem to be controversial but some researchers claim to obtain results that are nearly as good as conventional neural networks. The submitted paper mentioned above actually claims better results in comparison to an LSTM. Their technique seems almost comparable to a random forest, with a majority vote of multiple ELMs used for classification. It would be interesting to compare such an ELM approach to a CNN based technique, as it shares some similarities (e.g., both are based on local structure) and some differences (e.g., ELM can use nonlinear activation functions while the convolution operator used in CNNs is linear).

     It would be interesting to implement and test an ELM based architecture. This architecture could be compared to a more standard CNN based approach. For test cases, we could consider the image-based malware detection problem, for example.

127. A paper that I recently reviewed proposes a fairly simple opcode graph based technique for metamorphic malware detection. It is claimed that the results provide some insight into the underlying metamorphic generators, as opposed to simply providing a statistical model. It would be interesting to try to achieve similar results based on machine learning (ML) models. That is, we would like to automatically extract characteristics of an underlying metamorphic generator, based on ML models trained on samples generated by the metamorphic engine. This could also be applied to malware "families" in general, whether they are strictly metamorphic or not. We could experiment with simple morphing strategies and analyze the resulting ML models to try to connect model characteristics to generator (or family) characteristics. HMMs might work well, and SVM feature analysis could be useful. Deep neural networks could prove particularly interesting as, in principle, they should contain a wealth of information that we can attempt to analyze to determine characteristics of morphing engines (or general families).

128. A recent student project compared Android malware detection using a variety of machine learning techniques based on two distinct sets of features (`https://scholarworks.sjsu.edu/etd_projects/700/`). One set of features was extracted using emulation,

while the other set of features was extracted directly from the software as it executed on the phone. The detection rates using on-phone features were only marginally better than those obtained based on emulated features, and hence it would seem that the extra effort required for on-phone analysis is not justified. However, a significant percentage of apps failed to execute on the emulator. It would be interesting to perform similar experiments, but using different tools to see if we can successfully extract features from a higher percentage of apps using emulation. We would also like to collect more informative features to see if we can improve the detection rates. We could use a more recent state-of-the-art emulator, such as DroidScope, and for the feature extraction we could use the tool CopperDroid, for example.

In both cases (emulator and on-phone collection) we would want to collect sequential information features, as opposed to the simple histograms collected in previous work. Sequential information should provide stronger detection results (a PHMM would be an ideal technique to make use of such sequential information). The previous work mentioned above showed that a simple majority vote based on multiple classifiers yielded significantly better results than any of the individual classifiers. In this project, we would want to experiment with a wide variety of such ensemble techniques.

129. One of my recent masters students collected 3-d accelerometer data from a smartphone for 80 users, where each user traced their own unique signature in the air 20 times. Each user was then shown the same specific signature (that of the student collecting the data) and asked to mimic it, which they attempted to do 20 times. We want to analyze this data to determine the potential for this type of "in air" signature as an authentication system. Specifically, we want to determine the fraud and insult rates.

    The masters student computed a couple of elementary stats and used an SVM and obtained reasonable results. But, really, her contribution was mostly in the data collection phase. We have a rough draft of a paper that summarizes some of these elementary results.

    Here's an idea for something interesting to do with this data.

    (a) For each sample of a user's 3-d data, we computer the first two principle components. Intuitively, this gives us the two (orthogonal) directions with highest variance.

    (b) Then we use these principal components to project each signature onto two dimensions in the direction orthogonal to the two dimensions determined by PCA. Intuitively, each such projection will give us a flat 2-d image that represents the letters of the in-air signature.

(c) Then, for classification, we'll apply CNN-based image analysis to these 2-d projections.

For example, if the "signature" drawn in the air is an "S", the process described above should project it to either a forward S or a mirror image S (at some rotation) in 2-d space. Dealing with the mirror image possibility might not be an issue—as long as all signatures of a given user are projected in the same way, it doesn't matter whether it's in the forward orientation, or the mirror image. Finally, the data is ready and seems to be in reasonably good shape, so that is not a major issue.

The goal of this project is to determine how well CNNs trained on these 2-d projections perform in terms of the insult and fraud rates.

130. A recent research paper discusses "experimental bias in malware classification across space and time" (`https://arxiv.org/pdf/1807.07838.pdf`). Bias "across space" simply refers to imbalance in the size of the training and test sets, as compared to real-world data (e.g., we often use an equal number of malware and benign samples, when in a realistic detection scenario, we generally expect to see far more benign samples than malware of the specific type that a model is designed to detect). This is a well-known problem.

The "time" bias refers to using samples that do not belong together in time. For example, we might use malware samples collected over several years, but benign samples from one narrow time window. In effect, this trains the malware model on "future" information that would not have been available at the time the model would have been used. This problem seems to not have been considered much (if at all) in previous research. The authors of the paper cited above show that such a bias can inflate detection results.

For this project, we want to look closer at the issue of "time" bias in malware detection. A student recently completed a project where she used linear SVMs to automatically detect points in time where a significant change occurs in a malware family (`https://scholarworks.sjsu.edu/etd_projects/708/`). Using this technique, we could split the samples for a malware family into different time windows, and thereby create subsets of each family that belong together in time. This technique should enable us to maximize the amount of data available for training and testing, while reducing any time bias. For this project, we would use this SVM-based technique to split malware datasets and then determine how well various machine learning techniques (and various features) perform when using samples without a time bias, as compared to samples that includes a time bias. We have access to a couple of large datasets—one for Windows malware and one for Android malware—that would be ideal for this research.

131. I recently reviewed a paper with the following abstract:

> Detecting new malware and their variants remains both an operational challenge and a research challenge. In recent years, there have been attempts to design machine learning techniques to increase the success of existing automated malware detection and analysis. In this paper, we build a modified Two-hidden-layered Extreme Learning Machine (TELM), which uses the dependency of malware sequence elements in addition to having the advantage of avoiding backpropagation when training neural networks. We achieve this goal by using partially connected networks between the input and the first hidden layer. These are then aggregated with a fully connected network in the second layer, and finally, we utilize an ensemble to improve the accuracy and robustness of the system for malware threat hunting. The proposed method speeds up the training and detection steps of malware hunting (compared with stacked Long Short Term Memory (LSTM) and Convolutional Neural Network (CNN)). It does this by avoiding the backpropagation method and using a more simple architecture. Hence, the complexity of our final method is reduced, which leads to better accuracy, higher Matthews Correlation Coefficients (MCC), and Area Under the Curve (AUC), in comparison with a standard LSTM and CNN with reduced detection time. Our proposed method is especially useful for malware threat hunting in safety-critical systems, such as electronic health or Internet of Military of Things. In these areas, the enormous size of the training data makes it impossible to use complex models (e.g. deep neural networks), since training and detection speeds and detection rate are equally important. Our research results in a powerful network that can be used for all platforms with a broad range of malware analysis. The proposed approach is tested on Windows, Ransomware, Internet of Things (IoT) and a mixture of different malware samples datasets. For example, evaluation findings from an IoT-specific dataset, our proposed method achieves an accuracy of 99.65% in detecting IoT malware samples, with AUC of 0.99, and MCC of 0.992, thus outperforming standard LSTM based methods for IoT malware detection in all metrics.

That is, the authors use a so-called Extreme Learning Machine (ELM) for malware detection/classification and claim to obtain better results as compared to more standard techniques, including CNN and LSTM. An ELM is, essentially, a neural network with random weights and no training of the hidden layers (i.e., only the output layer is trained, and usually in a single step). Since no backpropagation is required, an ELM can be trained thousands of times faster than a traditional neural network. ELMs are somewhat controversial in the ML world.

I'm skeptical of the authors' claim that their ELM-based technique (which is actually an ensemble of multiple two-hidden-layer ELMs, or TELM) is superior to, say, an optimized CNN. For this project, we would want to compare an approach similar to the authors TELM to a CNN (and other standard techniques) over a large malware dataset, and using a variety of features.

132. Another possible project involving ELMs (see topic number 131, above) would be in the area of Real-Time Machine Learning (RTML). In RTML, we want to update models in real time. Usually, this involves using much simplified models, but another options is to train HMMs using a gradient ascent technique (instead of the more common Baum-Welch re-estimation). A good project would consist of comparing the results obtained using such an HMM approach to one based on ELM for an RTML problem. We might also want to consider other, more standard RTML techniques. We could test this on malware by, for example, generating new models over a sliding window of time. This would make sense in cases where the malware evolves in some significant way; see this recent project for a discussion of malware evolution: `https://scholarworks.sjsu.edu/etd_projects/708/`. Other RTML problems could also be considered.

133. Recurrent neural networks (RNN) are a powerful technique for dealing with sequential or time series data. However, RNNs suffer from the vanishing gradient problem, which severely limits the effective time window that we can consider when training and classifying with such models. Long short-term memory (LSTM) models include an internal "cell" state that, in effect, allows the model to store longer-term information, and thus is an effective workaround for the vanishing gradient. LSTMs have proven very effective in practice, yet they are complex, not well understood, and require substantial training data. Many variations on LSTMs have been considered (GRUs, are one such example), yet most do not perform significantly better than LSTMs.

    In this project, you will study a variant of LSTMs that will use an HMM to store long term information. That is, we will replace the cell state of an LSTM with an HMM, and we'll refer to the resulting model as a long short-term hidden Markov model (LSTHMM). There are actually several possible ways to embed an HMM in an LSTM, and you will experiment with such architectures. Such an approach should be able to provide richer long term information to the model, yet it will be easier to train. As far as I am aware, this is an entirely novel approach.

134. Here is a recent paper, titled "Decoupling coding habits from functionality for effective binary authorship attribution": `https://content.iospress.com/articles/journal-of-computer-security/jcs191292` It looks like this work might also be relevant to the problem of piracy detection—see topic 59, above, and my previous students' papers at `https://www.tandfonline.com/doi/abs/10.1080/19393555.`

`2013.787474` and, especially, `https://www.tandfonline.com/doi/abs/10.1080/19393555.2014.975557` In this previous work, we consider software piracy, where someone modifies the software in an effort to disguise their piracy.

This project would consist of implementing techniques analogous to those in the "authorship attribution" paper, and apply this (together with various machine learning and/or deep learning techniques) to the software piracy problem. The goal is to develop a technique that is more robust than previous work, in the sense that we can distinguish pirated software, even after extensive modifications have been made to obfuscate the source of the software.

135. Word2Vec is a word embedding algorithm based on a shallow (one hidden layer) neural network. The Word2Vec word embeddings are based on the weights of the hidden layer that are obtained when training the model. Note that in Word2Vec, the model itself is not ultimately used—it is simply the weight vectors that are obtained during training that are of interest.

It would be interesting to experiment with a similar approach based on an HMM—call it HMM2Vec. If we train an HMM on words, with $N$ hidden states, then the columns of the $B$ matrix in the trained model gives us vectors of length $N$ corresponding to the $M$ distinct words in our training set. These vectors "encode" some information about the words and should be useful in a similar way that Word2Vec encodings are useful. It would be interesting to compare the word encodings obtained via HMM2Vec to those obtained using Word2Vec.

An HMM is typically based on a Markov model of order one. In such a case, the HMM2Vec word encodings will be based on adjacent words. It would also be interesting to try to generalize HMM2Vec so that it includes greater context (similar to Word2Vec), rather than just being based on adjacent words.

Another option for generating embedding vectors is to apply PCA to a matrix of pointwise mutual information (PMI). To construct a PMI matrix, based on a specified window size $W$, we compute $P(w_i, w_j)$ for all pairs of words $(w_i, w_j)$ that occur within a window $W$ of each other within out dataset, and we also compute $P(w_i)$ for each individual word $w_i$. Then we define the PMI matrix as

$$X = \{x_{ij}\} = \log \frac{P(w_j, w_i)}{P(w_i)P(w_j)}$$

We treat column $i$ of $X$, denoted $X_i$, as the feature vector for word $w_i$ Next, we perform PCA (using the SVD) based on these $X_i$ feature vectors, and we project the feature vectors $X_i$ onto the resulting eigenspace. Finally, by choosing the $N$ dominant eigenvalues for this projection, we obtain embedding vectors of length $N$.

It is has been claimed that PCA2Vec embedding vectors have many similar properties as Word2Vec embeddings. Interestingly, it is also claimed that it can be beneficial in certain applications to omit some of the dominant eigenvectors when determining the PCA2Vec embedding.

136. To train an extreme learning machine (ELM), we randomly initialize the weights of the hidden layer, then we train the weights of the output layer (using a simple least squares technique). This makes training extremely fast and efficient, and will typically require less training data, as fewer parameters need to be learned. This recent master's project compares ELMs to CNNs in the context of malware detection: `https://scholarworks.sjsu.edu/etd_projects/900/`

    In analogy to an ELM, let's define an extreme Markov model (EMM) as follows. An EMM is the same as an HMM, except that the $A$ matrix is assigned (either at random, or based on knowledge of the problem), and is not re-estimated. The $B$ matrix is trained. Analogous to an ELM, an EMM will be faster to train (although only slightly, if we use Baum-Welch) as compared to an HMM, and an EMM will require less data for training.

    It would be interesting to experiment with EMMs to determine how effective they are, in comparison to a standard HMM. Also, ensembles of EMMs and/or the best EMM (as determined by the model score) based on a series of random restarts could be compared to the analogous result with HMMs. We could start with simple problems, such as English text analysis (using $N = 2$ hidden states), simple substitution cryptanalysis, and so on. We could also consider more complex problems, such as malware analysis based on various features, such as opcodes or $n$-grams.

    In effect, an ELM uses a random function for its hidden layer, while an EMM uses a random *linear* function as its hidden layer. We might be able to gain some insight into ELMs in general, based on our results for EMMs, since the EMM structure is inherently simpler and easier to analyze.

    Finally, when training an EMM, we might be able to use a different (and faster) technique than Baum-Welch.

137. There have been several recent papers where executable files are converted to images, and then image analysis techniques are applied to classify the images. Many techniques (SVM, CNN, ELM, and transfer learning, among others) have been applied in this context. A few of our recent papers focus on malware detection based on images—examples of which include the following:

    - `https://scholarworks.sjsu.edu/etd_projects/900/`
    - `http://www.cs.sjsu.edu/faculty/stamp/papers/vikash.pdf`

And there are some recent papers that consider GANs in relation to the malware problem:

- `https://arxiv.org/pdf/1702.05983.pdf`,
- `https://www.cs.tufts.edu/comp/116/archive/fall2018/tklimek.pdf`, and
- `https://ieeexplore.ieee.org/document/8669079`

For this project, we'll convert executable files to images, and then apply GANs for classification. This will be based on large malware datasets that we have available, with a large number of families (20 or 25) and a large number of samples per family (at least 1000).

138. In this project, we want to test the ability of generative models to produce malware that can defeat machine-learning based detection schemes. A simple generative model that we can easily test is an HMM. Suppose that we train an HMM on opcodes extracted from malware samples. We can then use the trained model to generate opcode sequences that match the malware samples used to train the HMM. We would then want to see how well we could distinguish these fake malware samples from real samples using various ML techniques. Similarly, we can use a GAN to generate fake malware samples, and perform similar classification experiments on the fake samples. Such experiments would enable us to determine not only the strength of the fakes, but they would allow us to test the strength and robustness of various detection strategies.

139. In a $k$-nearest neighbor ($k$-NN) classifier, values of $k$ that are "too small" tend to result in overfitting. For example, if we choose $k = 1$, then the training accuracy is necessarily 1, but the test accuracy is, in general, not 1, and it could be significantly less than 1. On the other hand, choosing $k$ "too big" will result in underfitting. A general rule of thumb seems to be that for the binary classification problem, we want $k \approx \sqrt{N}$, where $N$ is the number of samples in the training set.

     Although perhaps somewhat less intuitive, a similar problem can occur in a random forest (RF), with respect to the (maximum) depth of the component trees—a depth that is too large will tend to overfit, whereas a depth that is too small will result in underfitting. An illustrative example is discussed in the "`max_depth`" section at this url:

     `https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb`

     At the end of Section 6.4 of the 2nd edition of my ML book, there is a brief discussion of the connection between $k$-NN and RF. The upshot is that both techniques are neighborhood-based classifiers, but with different neighborhood structures. The purpose of this project is to draw out these connections, both anlaytically (i.e., in terms of the formulas that define the neighborhoods) and empirically (i.e., in terms of experiments). For the empirical part, the formulas in the ML book will be the starting point. For the experimental part, we want to consider binary (and possibly, multi-class) classification, based on a large dataset. The goal is to show that the experimental results align reasonably well with results of the analytic results, and to give meaningful experimental results that illustrate the pitfalls of overfitting and underfitting with these particular classification techniques.

Since we have extensive malware datasets, we would probably want to consider malware for the empirical results. And, in addition to binary classification, we would want to extend this to the multi-class case.

140. In some machine learning applications, we have a highly imbalanced dataset, typically with a small number of positive samples in comparison to a relatively large number of negative samples. This can cause problems when trying to train and test models. One possible solution is to augment the positive dataset with synthetic (i.e., fake) data. In practice, such synthetic data is often generated using ad hoc means.

A better approach would be to use machine learning to the greatest extent possible. If we can train a generative model on our limited dataset, then we can use the model to generate more data. A trained HMM is an example of a generative model, since it can be generate sequences by simply generating random numbers and consulting the $\pi$, $A$, and $B$ matrices to produce observations. From the perspective of the HMM, these fake sequences will look the same as the sequences on which it was trained. For example, if we train an HMM on English text, then we can generate fake English text that the HMM itself cannot distinguish from real English text. From a human perspective, however, the fake text would look fake.

In fact, we have used HMMs to generate synthetic data for various research problems. But, this approach is somewhat limited, since the synthetic data is so closely tied to a single model. It might, therefore, be better to vary the trained model parameters somewhat when generating fake data. We would like to automate the process as much as possible, so we need a systematic way to manipulate the model parameters.

In this research, we propose investigating various combinations of a generative machine learning algorithm with a genetic algorithm—the generative algorithm will be trained on the available data, and the parameters of the resulting model will be manipulated using a genetic algorithm. There has been some previous research involving the use of genetic algorithms to generate fake data, see, for example, `https://dl.acm.org/doi/10.1145/2970030.2970034`.

Specifically, we would probably want to start with HMMs trained on malware datasets. Then we would experiment with genetic algorithms to modify the HMM parameters when generating synthetic data. Finally, we would validate the quality of our synthetic data using other ML techniques. For example, we might train an SVM on the real data, and see how well it could distinguish real from the HMM-generated fake data. Another test could consist of training a model on both the real and fake data, then testing it to see if this data-augmented model is stronger than one trained only on real data.

We would want to experiment with other other generative ML techniques and, possibly, with non-malware datasets as well. This technique would likely be especially powerful if we could combine multiple generative techniques into one generator.

141. Recently, there have been 2 excellent master's projects that used machine learning to detect malware evolution: A paper based on Mayuri Wadkar's project is available at `https://www.sciencedirect.com/science/article/abs/pii/S0957417419307390`, and a paper based on Sunhera Paul's project is available from me, on request. In both cases, the main tool was linear SVMs. Specifically, linear SVMs are trained at one month offsets, using the previous year of samples as the positive class and the next month of samples as the negative class. Then a $\chi^2$ statistic is computed based on the SVM weights at adjacent offsets. We plot this $\chi^2$ statistic over a period of time, and any large spikes in the resulting graph indicate that the malware samples used for training the corresponding models differ. Such spikes are possible points of significant evolution. We considered a variety of features in this previous research, including a set of PE file features, opcode sequences, and word embedding (e.g., Word2Vec) based features.

    In the most recent of these 2 projects, we used HMMs to further confirm (or deny) that evolution had taken place at each spike in the $\chi^2$ graph. To do so, we train an HMM on a subset of samples before the spike, and we use this HMM to score a distinct subset of samples from before the spike, as well as samples after the spike. If the resulting score distributions differ significantly, this provides strong evidence that the we have found a major evolution point in the timeline of the malware family under consideration. For additional confirmation, we can repeat this process, training another model on samples after the spike.

    The purpose of this project is to further analyze the HMM technique discussed in the previous paragraph (and related techniques). Specifically, we want to determine whether we can detect malware evolution using this HMM approach, without employing the initial SVM phase. We will compute an HMM for each time window, at monthly offset. Then at a given point in the timeline, call it $X$, we compare the models trained in data ending at $X$ with the model trained on data that begins at time $X$. We can then use both of these models to score samples from the month before and the month after $X$, as discussed above. Another option is to directly compare the HMM models—there are several techniques for measuring the "distance" between HMMs.

We would want to experiment with different features (e.g., opcodes and byte $n$-grams). We might also consider different ML techniques or combinations of techniques. For example, we could first use HMMs to detect potential evolution points, then use SVMs to confirm (or deny) such points, which is the opposite of the order considered in the previous work discussed above. It would also be interesting to experiment with various types of neural networks, including Word2Vec, which could be used in place of the HMM.

142. A recent paper considers Android malware detection based on the use of Gabor filters applied to images generated from `dex` bytecode files. Gabor filters are used for textural analysis of images; there is a pretty good description here: `https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97`

     Wrt malware, this work is interesting since it seems to be able to detect similar regions, as opposed to just the overall characteristics of the samples. A more thorough analysis applied to a larger and more diverse malware dataset would be a good project topic.

143. In a generative adversarial network (GAN) two competing networks are trained simultaneously. We can do a somewhat similar thing when training an HMM. Suppose that the training data is limited. Then we can perform a large number of random restarts. Another option might be to do a small number of random restarts, then use the best model in a generative mode to produce more training data, training another model on this expanded dataset. This process of augmenting the data and retraining could be repeated many times. We would need to experiment to decide if this offers any advantage over random restarts. In analogy to a GAN, we could refer to this techniques as a GAHMM.

144. An inherent weakness of hill climb algorithms is that we only find a local maximum. One way that we can attempt to overcome this limitation is by using multiple (random) restarts, that is, we run the algorithm multiple times, with different initial conditions each time. The Baum-Welch algorithm that is used to train an HMM is a hill climb technique, and random restarts are often used when training HMMs. Since the Baum-Welch algorithm produces a model score, we can easily determine the best model from among multiple models.

     One generic alternative to random restarts for hill climb algorithms is to use "momentum" to intentionally overshoot a local maximum. For example, if a hill climb tells us to increment a parameter $P$ by $x$, we can instead increment $P$ by $x + \varepsilon$ for some small value of $\varepsilon$. In this way, we may be able to overcome a local maximum to obtain a better result.

It would be interesting to modify Baum-Welch to use momentum. Specifically, when we update the matrices, we can add a small increment to each element $a_{ij}$ of $A$ and $b_j(k)$ of $B$. Of course, we need to then make sure that the resulting matrices are row stochastic, but this is easily accomplished by dividing each element by its row sum.

We would want to compare HMMs with random restarts to HMMs with momentum on some representative problems. Substitution ciphers would be a good test case, since we obtain much better results using a large number of random restarts in cases where the ciphertext is limited. We could also experiment with malware detection or classification.

145. A recent paper considers the question of whether behavior-based malware detection is more focused on malicious behavior or evasive behavior. It is claimed that evasive behavior is what is actually detected, at least for the one type of (simulated) malware considered. However, their results did not seem persuasive as there were many weaknesses in the paper. But, the problem is interesting, and it could make a good research project. We would need to be able to extract features that could be categorized as either malicious or evasive, then train on such features, and finally perform feature analysis to determine which type of features are most relevant. Experiments would need to be conducted for each of several different families, preferably from different classes of malware. In the paper mentioned above, system calls are used as features, but other features might be considered. This topic might be especially interesting for Android malware.

146. In machine learning and (especially) deep learning, models are often opaque, meaning that it is unclear how decisions are being made. There have been cases where models give accurate results, but they are, in effect, "cheating" and this was only discovered by looking closely at how the models were making decisions. An example of a model that was discovered "cheating" is discussed here: `https://techcrunch.com/2018/12/31/this-clever-ai-hid-data-from-its-creators-to-cheat-at-its-appointed-task/`

Thus, it is important to look at the inner workings of models to determine how they are making decisions. One generic approach is to consider "counterfactual" explanations. For example, suppose that we are considering a binary classification problem, with class labels $\{+1, -1\}$. Then given a sample $X$ with classification $z = f(X)$, a counterfactual example is an $X'$ such that $f(X') = -z$, for which the "distance" between $X$ and $X'$ is, in some sense, minimal. By considering such "nearby" pairs that are classified differently, we can obtain some insight into how the model is making its decisions.

This article discusses several generic approaches to finding counterfactual examples: `https://christophm.github.io/interpretable-ml-book/counterfactual.html`.

An even simpler approach would be the following. Consider again a sample $X$ with $f(X) = z$. We could simply find the "nearest" sample $X'$ in our training set such that $f(X') = -z$. This $X'$ would not necessarily be minimal distance over all possible samples, but it might be good enough in some cases.

With some machine learning algorithms, it is possible to explicitly determine counterfactual examples. For example, for a linear SVM we can find examples that are just across the decision boundary.

For this project topic, we want to focuses on counterfactual observation sequences for a given HMM. Your instructor has has written a brief paper describing a few ways that we might construct such examples—if you are interested in this topic, ask for a copy of the pdf.

147. I recently reviewed a paper that was focused on detecting "darknet" traffic. While the paper itself was not too interesting, they did have a nice dataset, which seems to be available from the authors of this paper (not the one I reviewed, btw):

    A. Berman and C. L. Paul, Making sense of darknet markets: Automatic inference of semantic classifications from unconventional multimedia datasets, in: *International Conference on Human-Computer Interaction*, Springer, 2019, pp. 230–248.

    It would be interesting to get this dataset and use some ML/DL techniques to determine how well we can detect such traffic. Some straightforward—and effective—techniques for malicious traffic detection are discussed in this paper `https://www.sciencedirect.com/science/article/pii/S0167404814000959`. We could experiment with similar techniques for darknet traffic, which could serve as a baseline for comparison with more advanced ML/DL techniques, including HMM, SVM, MLP, LSTM, etc.

148. There has been some interesting recent work focused on identifying the smartphone camera used to take a given photo (or video) based on various noise artifacts; see, for example, the paper `https://hal.archives-ouvertes.fr/hal-01915647/file/ESPIC_Camera_Identification_JPEG_HAL.pdf`.

    It would be interesting to see if we can generate "deep fake" images (or video) that mimic the identifying noise information of a specific camera. If so, this would result in the ability to generate fake images that appear to have come from a specific (smartphone) camera. GANs would seem to be the proper tool for this job.

    Finding data for this project could be challenging. But, if such data is available or can be collected, this research problem is a very interesting one.

149. In transfer learning, we adapt a pre-trained model to new data, typically by retraining the output layer, while leaving the hidden layers unchanged. The paper `https://papers.nips.cc/paper/2019/hash/fd2c5e4680d9a01dba3aada5ece22270-Abstract.html` discusses a technique that they refer to as "TransNorm" that is claimed to improve transfer learning, by adapting the features so that they "transfer" across the different domains (i.e., the original data that the model was trained on versus the new data that will be used for transfer learning). The authors claim to obtain improved results on a variety of problems using TransNorm in combination with various deep neural network architectures.

     It would be interesting to study TransNorm in transfer learning application in the malware domain. In these papers, we experiment with transfer learning for image-based analysis of malware `http://www.cs.sjsu.edu/faculty/stamp/papers/vikash.pdf`, `https://arxiv.org/abs/2103.13827` A good project topic would consist of combining such image-based analysis with TransNorm, to see if we can obtain improved results.

     The Malware Capture Facility Project (MCFP) includes network traces that involve malware—see `https://www.stratosphereips.org/datasets-malware` and `https://mcfp.weebly.com`. We could perform image-based analysis on this data, and compare the results obtained using standard transfer learning techniques to those obtained using TransNorm.

150. Recently, there have been a few papers where people embed malware in neural networks. The idea is that such models provide a way to distribute malware, which, in the ideal case, could be triggered by a specific input. The goal is to embed the malware in such a way that it has minimal negative effect on the functioning of the model. An example of such research can be found here:

     `https://dl.acm.org/doi/10.1145/3427228.3427268`

     Such malware can be viewed as using a form of steganography, which (in the paper cited) relies on redundancy in insignificant bits in the weights to embed information. A classic example of steganography is embedding information in the low-order bits of a color (RGB) image. Such information has no visible effect on the image, since the low order bits are essentially redundant. In terms of weights in deep learning models, we do not require great precision, so bits far to the right of the decimal point can be used to hide information.

     It would be interesting to consider steganographic machine learning (stego-ML) malware attacks in various contexts. For example, we could start with simpler models, such as HMMs and SVMs, to see if they are subject to such attacks. It might be the case that stego-ML attacks are more challenging with classic models, as compared to neural networks, since there may be less redundancy in classic models. If so, this would indicate that classic models have a security advantage that might be relevant in high-security applications.

In any case, it would be interesting to use some specific aspects of ML/DL models for steganography. That is, in previous work, information is hidden in redundant bits of weights, which is a technique that is applicable to any floating point numbers. In terms of DL models, perhaps we could select weights that have little or no influence on the model, and use more of the bits in those specific weights to hide information. In a linear SVM, for example, the small weights contribute little to classification, and so we should be able to use those for hiding information; in an HMM, the $\pi$ matrix is usually not crucial, so perhaps we can use it for information hiding; in an LSTM perhaps we can hide information in the cell state, and so on.

Another line of research would be to simply determine the "capacity" of various types of ML models. For example, how many bits are "free" for steganography in an HMM of a specified configuration, an SVM, MLP, etc. Considering pre-trained models (VGG19, ResNet50, etc.) would also be interesting. One approach to this would be to experiment by hiding info in low-order bits positions, and determine the tradeoff between the number of bits used for information hiding versus the loss in accuracy of the model. More generally, we would like to minimize the redundancy (i.e., minimize the number of bits needed to represent a model) which would minimize the ability to hide information in a model.

Ways to trigger malware that is embedded in ML models would be another interesting—although, likely very challenging—aspect to this topic.

151. Suppose that we use, say, a fingerprint biometric for authentication. In such a system, for user Alice, we store some statistical information extracted from a scan—or multiple scans—of Alice's fingerprint. Then, when someone claiming to be Alice wants to authenticate, we extract the same type of statistical information and feed it into a model to determine whether it matches the data stored for Alice, or not.

Suppose that a bad "guy," Trudy, gets ahold of the file that we use for authentication—the file that contains Alice's statistical fingerprint-derived information. Can Trudy use this information to generate a fake fingerprint that will enable her to authenticate as Alice? We good guys would prefer that the answer is "no," since the system would then be "one way," analogous to a cryptographic hash functions that is used for storing passwords. In fact, a "one way" biometric system would be even stronger than hashing passwords, since the forward search attack is likely to be far more difficult for most biometrics.

This question can apply to any type of biometric, and we can even consider whether ML/DL techniques themselves are "reversible." A recent paper considers the case of a palmprint biometric, and shows that sufficiently accurate fake palm prints can be generated from the information stored in the authentication file.

We have recently done some work where "signatures" written in the air with a smartphone are used for biometric authentication. It would be interesting to consider whether this biometric technique is reversible. Fingerprints or other biometrics could be considered.

As mentioned above, we could also consider this problem from the perspective of ML/DL techniques. For example, if we have CNN that distinguishes between images of cats and dogs, can we use the model to directly generate images that the model will then classify as cats and dogs? We are not concerned with whether the images appear to humans to be cats and dogs, just whether the model can be fooled by such fake images.

152. A recent paper considers "backdoor" attacks on neural networks, that is adversarial attacks where a specific input triggers a specific (incorrect) output. For example, a specific image of a cat might be misclassified as a dog, while the classifier works well, in general, when classifying dogs and cats. They claim to be able to detect the presence of such attacks using a measure that they refer to as "wobbliness", which is based on a straightforward entropy calculation. They claim that the decision boundary is unusually smooth (i.e., less "wobble") near such adversarial attacks, which makes some intuitive sense, as most of the features don't matter in the same way that they would if a sample is classified normally.

There are lots of potential projects related to this "wobbliness" measure. For example, the authors only consider ResNet architectures, and we could apply this to specific attacks on other classifiers (a linear SVM, in particular, would offer perhaps the simplest test case and might result in interesting insights). We could also consider higher-level attacks, such as adversarial attacks designed to create lots of smoothness (equivalently, low wobbliness), without causing classification errors. This would result in many false alarms, if wobbliness is used to detect attacks. In this way, an actual attack might go unnoticed.

More generally, we might measure the wobbliness of different types of classifiers to see which are more similar according to this measure (e.g., we might expect $k$-NN and random forest to be similar, as well as SVM and MLP, while, for example, $k$-NN and SVM might be inherently different). We might also be able to apply this measure to the "explainability" problem, where we try to understand what a classifier is using to make its decisions—in areas of relatively low wobbliness, the decisions are simpler, in the sense that fewer features are relevant, whereas in areas of high wobble, the opposite would be the case. Perhaps, wobble could be combined with LIME, as areas of low wobble could be modeled by simpler (local) models, while high wobble would indicate areas that require more complex models.

153. The KronoDroid dataset is discussed in this article

`https://www.sciencedirect.com/science/article/pii/S0167404821002236`

This dataset consists of large number of Android malware samples, with many features extracted, making it particularly useful for ML/DL experiments. The dataset also includes a creation date for each sample, which is necessary for malware evolution experiments. A good project would consist of conducting such evolution experiments to automatically detect points in time where evolution has occurred. In a series of previous papers, we have found that linear SVMs work well for detecting evolution. By comparing linear SVM weights for models that have been trained over successive time windows, we find that changes in the SVM weight vectors (as measured by cosine similarity) reflect changes in the samples.

Then, once such "concept drift" (i.e., evolution) points have been detected, we would want to compare the following three cases for malware detection and/or classification.

i) We train one model on the initial time samples of a given family, and then use that model to classify all remaining samples in the family.

ii) The same as i), except that we update the classifier model whenever a sufficient level of concept drift has been detected.

iii) The same as ii), except that the updates occur regularly (e.g., every month), regardless of whether any concept drift has been detected.

We expect that ii) and iii) will be more effective than i), and that ii) will be more efficient than iii).

154. So-called extreme learning machines (ELM) are feedforward neural networks with a single hidden layer. In an ELM, hidden layer weights are assigned at random and are not updated. The training phase consists of modifying the weights of the output layer, which can be accomplished by, essentially, solving linear equations, and hence training is extremely fast (which, presumably, is the reason for the "E" in ELM).

This paper discusses the concept of an iterative ELM (I-ELM):

`https://ieeexplore.ieee.org/document/8712783`

The goal of this project is to develop and test various hill climb modification to the standard ELM algorithm—we will refer to these as HC-ELM algorithms. In the simplest case, we could specify a learning rate parameter $\alpha$ and iteratively modify each hidden layer weight $w$ as $w + \alpha$. After each such modification of a hidden layer weight $w$, we would recompute the output layer weights using the standard ELM training technique, and then compute the loss (or accuracy) of this modified ELM. If the modified weight improves the model (as measured by the loss or accuracy), we would retain the modified weight $w+\alpha$ and the newly-computed output layer weights; otherwise, we would leave the value of $w$ (and the output layer weights) unchanged for the next iteration.

There are numerous possible extensions to the simple HC-ELM in the previous paragraph. At each iteration, we could flip a coin and choose to test either $w+\alpha$ or $w-\alpha$, or we could test both. In addition, we would probably want to do more than one pass through the set of hidden layer weights. To accomplish this, we could define a second parameter $N$, which would be the number of "epochs", where an epoch refers to one pass through all of the hidden layer weights (as opposed to the usual definition of epoch, which refers to one pass through all of the training samples). Even better, we could use an algorithm similar to Jakobsen's hill climb approach, where we would return to the start of the list of hidden layer weights each time the model improves, and continue until we test the final weight on the list without finding any improvement in the HC-ELM model.

We would want to test the various HC-ELM strategies discussed above on a standard, well-studied problem—we have several malware datasets that could serve as such a testbed, but other problems domains could considered. For each proposed HC-ELM, we would want to compare the results to a standard ELM, and to a model trained using backpropagation. The goal is to find an HC-ELM approach that improves on a standard ELM, yet has a work factor that remains considerably lower than using backpropagation. Such an HC-ELM technique would be very useful in practice.

Finally, and more speculatively, it would be interesting to consider a Baum-Welch type of hill climb strategy to modify the hidden layer weights of an ELM (recall that Baum-Welch is the hill climb algorithm that is used to train an HMM). If such an approach is possible, we might call it HM-ELM, and it would likely be much more effective—and possibly much more efficient—than the hill climb strategies discussed above.

155. There has been a lot of talk recently to the effect that AI will become so smart that robots will take over the world. This is mostly based on the successes of deep learning models. However, there is some pushback, with several researchers claiming that current learning models are not capable of anything more than statistical discrimination, which is often stated (in a derogatory sense) as "curve fitting." The goal of this project is to see if a learning algorithm can solve a problem where some cleverness is required, as opposed to just a straightforward statistical discrimination problem. To do so, we will train a model on data derived from a problem where we know a "correct" answer. It is important that the data is suitable for training

learning algorithms, and that we judge that significant intelligence was required to obtain the known correct answer.

There are many problems that could be used as the basis for this project. But, since we are CS people, and CS is largely the study of algorithms, we will consider data obtained from a known algorithm. To state the problem succinctly, we want to address the following question: Can a learning algorithm learn an algorithm?

Which algorithm should we consider? One possibility is the Berlekamp-Massey algorithm, which computes the so-called *linear complexity* of a binary sequence, that is, the length of the shortest linear feedback shift register (LFSR) that can generate the sequence. We can easily create an endless supply of training and testing data consisting of matched pairs $(s, L)$, where $s$ is a binary sequence, and $L$ is its linear complexity. We will then train various learning models (SVM, MLP, etc.) on the training data, and test the resulting models on the test data to determine the accuracy of the trained models. The accuracy of a model is based on how often it correctly predicts the linear complexity of the sequences in the test set.

If a model can accurately predict the linear complexity, then the model has, in effect, learned something that is equivalent to the Berlekamp-Massey algorithm. Since this algorithm required a great deal of cleverness to derive, we can argue that the learning algorithm has some degree of cleverness. On the other hand, if learning models cannot be trained to accurately predict the linear complexity, then it calls into question whether they can do much of anything other than "curve fitting."

I have written some additional notes on this topic. If you are potentially interested, I can give you the pdf.

156. I recently reviewed a very nice paper that discusses "model stealing" attacks against malware detection system. That is, the authors discuss black-box attacks that enable them to determine ML models that perform similar to a specific malware detection system—they even test their approach against real antivirus products. A unique challenge to this type of problem in the malware domain is that a useful antivirus must have a low FPR.

    In this same paper, the authors then go on to show that they can use their stolen models to produce malware samples that evade detection by the corresponding original system. An interesting twist on this problem would be to try to use the stolen models to generate false positives, that is, benign samples that are misclassified as malware. Since useful antivirus must have a low FPR, if we can generate such benign samples, it could have a devastating effect on the viability of an antivirus product. Of course, we could also look at this from a more positive perspective, as such false positives would enable an AV vendor to improve their product.