GSWC 2012

Proceedings of

The Seventh Annual Graduate Student Workshop on Computing

October 5th, 2012
Santa Barbara, California

Department of Computer Science
University of California, Santa Barbara
http://www.cs.ucsb.edu

Organized By

Adam Doupé, Chair

Gianluca Stringhini, Vice-Chair

Mariya Zheleva, Industry Liaison

Ana Nika, Financial Coordinator

Nazli Dereli, Proceedings Coordinator

Sean Maloney, Web Coordinator

Dhilung Kirat, General Committee

Adam Lugowski, General Committee

Ali Zand, General Committee

Yun Teng, General Committee

Lara Deek, General Committee

Stephen Gauglitz, General Committee

Vineeth Kashyap, General Committee

Hassan Wassel, General Committee

Siladitya Dey, General Committee

Jane Iedemska, General Committee

Arijit Khan, General Committee

Thanks to

Platinum Supporters

Silver Supporters

Bronze Supporters

Keynote Speaker

Gus Class, Developer Advocate, Google

Gus is a developer advocate for the Google+ platform. Before joining Google, he worked on Microsoft Windows, Zune, Xbox, and developed anti-spam and web-based email software. You can find out more about him on Google+ at http://gusclass.com/+

Discussion Panel

Luca Bertelli, Software Engineer, Google

Luca received the D.Ing. degree (summa cum laude) in electronic engineering from the University of Modena, Italy, in 2003, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California, Santa Barbara, in 2005 and 2009, respectively.
During the summer of 2008, he was an intern at Google Research, working on salient object detection. After graduating he joined Mayachitra and later moved to Like.com, always working on computer vision related problems.
He is now at Google where he is a tech lead within Google Shopping, working on making your online shopping experience as amazing as possible!

Chad Sweet, Software Engineering Manager, Qualcomm

Chad Sweet has been with Qualcomm for more than 14 years and is currently leading a project to apply biologically inspired neural networks to robotics applications in Corporate R&D. He has a BS in Computer Engineering from Vanderbilt University, and has a number of patents granted or pending in the area of wireless technology.

Kevin Haas, Principal Development Manager, Bing Social, Microsoft

Kevin Haas earned dual Bachelors degrees in Computer Science and Mathematics from UC Santa Barabara, and a MS from Stanford University, where he studied databases and operating systems. Since graduating, he's held a variety of software development and management positions at IBM, Yahoo, and Microsoft. Currently, he leads Bing Social's engineering team, and responsible for the integration of data from Facebook, Twitter, and other social networks; and dabbles with unstructured text analysis and large-scale entity graphs on the side.

Table of Contents

Posters

Jarhead

Johannes Schlumberger

University of California, Santa Barbara js@cs.ucsb.edu

I. Introduction

Java applets have increasingly [3], [4] been used as a vector to deliver drive-by download attacks that bypass the sandboxing mechanisms of the browser's Java Virtual Machine and compromise the user's environment. State-of-the-art approaches to the detection of malicious Java applets are based either on simple signatures or on the use of honeyclients, which are both easily evaded. Therefore, we propose a novel approach to the detection of malicious Java applets based on static code analysis.

Java Applets usually come packaged as archives (called Jar files). To protect against malicious applets, the JVM sandboxes an applet and heavily restricts its permissible operations when it is not loaded from disk.

A sandboxed applet cannot access client resources, such as the file system. By restricting the abilities of untrusted mobile code, its abilities to infect an end user's machine or to tamper with her data are severely limited.

Malicious applets try to escape the sandbox and install malware on a victim's computer. To achieve this, some applets try to trick careless users into trusting their certificates. Others target the JVM itself by trying to exploit a vulnerability in the Java plugin, effectively disabling the sandbox and turning the applet into a full-fledged, non-restricted program with permissions equal to that of the user running the browser.

II. Jarhead System Overview

Jarhead relies on static analysis and machine learning to detect malicious Java applets. Jarhead analyzes the bytecode that is part of a class file. To analyze a Jar file, we extract its contents and then disassemble it.

Jarhead operates in two steps: First, we use a training set of applets - each applet known to be benign or malicious - to train a classifier based on the features Jarhead extracts from applets. After the training phase, we use this classifier in a second step to perform detection of malicious applets.

Java bytecode was specifically designed to be verifiable and easy to parse. Static analysis, therefore, works well on bytecode, and does not suffer from a lot of the limitations common to other binary program formats, such as computed jumps. Furthermore, a Java program does not have the ability to dynamically generate code at runtime, which is usually a problem for static analysis techniques. Thus, even when attackers make use of code obfuscation [5], a static analysis approach (such as the one proposed in this paper) can obtain sufficient information to correctly classify an unknown program (as our results demonstrate).

III. Feature Discussion

Jarhead collects a total of 42 features: Six are numeric, ten are integers, and the remaining 26 are Boolean values. These features can be divided into two categories. The *obfuscation features* and the features aiming at exposing the purpose of an applet by statically examining its potential behavior - the *behavioral features*.

a) Obfuscation Features: Obfuscation is an important aspect for all malware today. Obfuscated code differs from other code because it is generated by obfuscation kits in an automated way. These kits chop up string constants and introduce additional statements or declarations without changing the semantics of the original code. We use different code metrics as features to find differences between obfuscated and non-obfuscated code such as dead code, obfuscated Strings or the use of reflection. We also check if there is functionality to load code at runtime or execute a scripting language. Of course, obfuscation alone is not sufficient to identify a Java program as malicious, since obfuscation kits are also used by benign code. However, while manually examining many Java applets collected from the Internet, we found that obfuscation is overwhelmingly used by malicious applets.

b) Behavioral Features: The overwhelming majority of applet malware has a specific purpose: Escaping the Java sandbox to infect the end user's machine and make it a member of a botnet. The features in this section aim at statically identifying this behavior. We check for interactions with the runtime environment and the Java security system. Furthermore Jarhead determines if a Jar has the potential to download files or launch programs.

By detecting all known possible ways for an applet to spawn a new process, we make it impossible for malicious applets to achieve this without triggering the corresponding detection feature.

Benign applets are usually written with the goal of displaying something to the visitor of a web page or to play audio. Typically, the files necessary to do so (media files, images, ...) are included in the Jar archive. If an attacker tries to mimic a benign applet, this will raise his costs per attack.

Finally we also compiled a set of five well known vulnerable components of the Java library. Exploits that target well-known, vulnerable Java library functions have to call these functions to be successful. Of course, these functions also serve a legitimate purpose. However, we found that they are rarely used by benign applets (and our other features help in making this distinction). Their rare usage probably also indicates that they were not well-tested in the first place and thus the vulnerabilities were overlooked. If more vulnerable

functions were to be added to this set later on, we would expect them again to be somewhat obscure and rarely used by benign applets. These few functions provide a way to break out of the sandbox without user interaction in vulnerable Java versions.

Abstaining from the use of these functions significantly decreases the chances of success for an attacker, since passive infection without user interaction becomes impossible.

If an attacker tries to evade the obfuscation features, he will more likely be caught by traditional signature-based detection systems.

Even if an attacker finds a way to evade the obfuscation features, in order to achieve his purpose of infecting end users, his malware will necessarily implement functionality to achieve this goal (break out of the sandbox and start malware on the victim's PC). By capturing this behavior, intrinsic to the attacker's goal, we make it difficult for malicious code to evade our analysis and still succeed.

IV. Evaluation

We evaluated our system to measure how well it performs in detecting malicious applets. For our experiments, we used two different datasets: A manually-compiled dataset of applets (the manual set), and a set of samples collected and provided to us by the authors of the Wepawet system [2] (the Wepawet set).

a) Results: Manual Dataset: The manual dataset contains samples from four different sources: Two online collection of applets (that offer a selection of Java applets for web designers to use in their web pages), an archive of mostly-malicious applets, and a number of manually collected samples for a total of 2,095 samples.

To obtain ground truth for building and evaluating our C4.5 decision tree classifier, we submitted the entire manual dataset to Virustotal [1]. Virustotal found 1,721 (82.1%) of the files to be benign and 374 (17.9%) to be malicious (we counted a sample as benign if none of the Virustotal scanners found it to be malicious and as malicious if at least one scanner found it to be malicious).

Using the results from Virustotal as a starting point, we built an initial classifier and applied it to the manual dataset. We then manually inspected all samples for which the results from our classifier and the results from Virustotal were different. In this process, we found that Virustotal has actually misclassified 61 (2.9%) applets. We manually corrected the classification of these 61 applets, and used the resulting, corrected classification as the ground truth for our dataset (with 381 malicious and 1,714 benign samples).

With ten-fold cross validation, our classifier only misclassified a total of 11 (0.5%) samples.

If we compare our detection results to the results produced by Virustotal, we see a reduction in the number of misclassifications by a factor of six (in spite of the fact that we used Virustotal to build our initial labels and hence, our ground truth might be biased in favor of Virustotal). Despite a few misclassified instances, we found that our system performs detection with high accuracy. In particular, some of the incorrect cases are arguably in a grey area (such as possibly benign applets that try to execute commands directly on the Windows command line and malicious applets that contain incomplete, half-working exploits).

b) Results: Wepawet Dataset: To further test our classifier on real-world data, we collected 1,275 Jar file samples from the Wepawet system.

Virustotal found 413 (32.4%) applets to be benign and 862 (67.6%) applets to be malicious. We then ran our classifier on this Wepawet set. Compared to Virustotal, we assigned the same verdict to 1,189 (93.3%) samples, while 86 (6.7%) samples were classified differently.

Manual examination of the instances with different classifications revealed interesting results, both for the false positives and the false negatives. For the false positives, we found that 19 of the 27 were clearly errors in the initial classification by Virustotal (that is, these 19 applets were actually malicious but falsely labeled as benign by Virustotal). That is, Jarhead was correct in labeling these applets as bad. Interestingly, one of these 19 samples was a malicious applet that was stealing CPU cycles from the user, visiting a web page to mine bitcoins. While we had not seen such behavior before, Jarhead correctly classifies this applet as malicious based on its interaction with the runtime environment.

We then inspected the 59 applets that Virustotal labeled as malicious (while Jarhead labeled them as benign). Four programs were partial exploits that did not implement actual malicious behavior. Nine were false positives by Virustotal. We do not consider these partial exploits (which are essentially incompletely packed archives) and the nine benign programs to be properly labeled by Virustotal.

The remaining 46 samples were actual malware. They were largely (96%) made up of two families of exploits for which we had no samples in our (manual) training set, which we used to build the classifier. To show that we can achieve better results with a better training set, and to demonstrate that our features are indeed well-selected and robust, we trained and tested a new classifier on the Wepawet dataset using ten-fold cross validation. For that experiment, we found a total misclassification count of 21 (1.6%).

V. Conclusions

We address the quickly growing problem of malicious Java applets by building a detection system based on static analysis and machine learning. We implemented our approach and tested it on real-world data. We also deployed our system as a plugin for the Wepawet system, which is publicly accessible. Our tool is robust to evasion, and the evaluation has demonstrated that it operates with high accuracy.

References

[1] Virustotal. http://www.virustotal.com.

[2] Wepawet. http://wepawet.iseclab.org.

[3] Mike Geide. 300% increase in malicious jars. http://research.zscaler.com/2010/05/300-increase-in-malicious-jars.html, 2010.

[4] Brian Krebs. Java: A gift to exploit pack makers. http://krebsonsecurity.com/2010/10/java-a-gift-to-exploit-pack-makers, 2010.

[5] Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, 2003.

Message In A Bottle: Sailing Past Censorship

Luca Invernizzi[1], Christopher Kruegel[1], Giovanni Vigna[1]

[1] UC Santa Barbara

***Abstract*—Exploiting recent advances in monitoring technology and the drop of its costs, authoritarian and oppressive regimes are tightening the grip around the virtual lives of their citizens. Meanwhile, the dissidents, oppressed by these regimes, are organizing online, cloaking their activity with anti-censorship systems that typically consist of a network of anonymizing proxies. The censors have become well aware of this, and they are systematically finding and blocking all the entry points to these networks. So far, they have been quite successful. We believe that, to achieve resilience to blocking, anti-censorship systems must abandon the idea of having a limited number of entry points. Instead, they should establish first contact in an online location arbitrarily chosen by each of their users. To explore this idea, we have developed Message In A Bottle, a protocol where any blog post becomes a potential "drop point" for hidden messages. We have developed and released a proof-of-concept application using our system, and demonstrated its feasibility. To block this system, censors are left with a needle-in-a-haystack problem: Unable to identify what bears hidden messages, they must block everything, effectively disconnecting their own network from a large part of the Internet. This, hopefully, is a cost too high to bear.**

I. Introduction

The revolutionary wave of protests and demonstrations known as the *Arab Spring* rose in December 2010 to shake the foundations of a number of countries (e.g., Tunisia, Libya, and Egypt), and showed the Internet's immense power to catalyze social awareness through the free exchange of ideas. This power is so threatening to repressive regimes that censorship has become a central point in their agendas: Regimes have been investing in advanced censoring technologies, and even resorted to a complete isolation from the global network in critical moments. To sneak by the censorship, the dissident populations have resorted to technology. A report from Harvard's Center for Internet & Society [2] shows that the most popular censorship-avoidance vectors are web proxies, VPNs, and Tor. These systems share a common characteristic: *They have a limited amount of entry points.* Blocking these entry points, and evading the blocking effort, has become an arms race: China is enumerating and banning the vast majority of Tor's bridges since 2009, while in 2012, Iran took a more radical approach and started blocking encrypted traffic, which Tor countered the same day by deploying a new kind of traffic camouflaging.

In this paper, we take a step back and explore whether it is possible to design a system that is so pervasive that it is impossible to block without disconnecting from the global network. Let's generalize the problem at hand with the help of Alice, a dissident who lives on the oppressed country of Tyria and wants to communicate with Bob, who lives outside the country. To establish a communication channel with Bob in any censorship-resistant protocol, Alice must know *something* about Bob. In the case of anonymizing proxies or mix-networks (e.g., Tor), this datum is the address of one of the entry points into the network. In protocols that employ steganography to hide messages in files uploaded to media-hosting sites (such as Collage [1]) or in network traffic (such as Telex [3]), Alice must know the location of the first rendezvous point.

The fact that Alice has to know something about Bob inevitably means that the censor can learn that too (as he might *be* Alice). Bob cannot avoid this without having some information to distinguish Alice from the censor (but this becomes a chicken-and-egg-problem: How did Bob get to know that?). We believe that this initial *something* that Alice has to know is a fundamental weakness of existing censorship-resistant protocols, which forms a crack in their resilience to blocking. For example, this is the root cause of the issues that Tor is facing with distributing bridge addresses to its users without exposing them to the censor. It is because of this crack that China has been blocking the majority of Tor traffic since 2009: the number of entry points is finite, and a determined attacker can enumerate them by claiming to be Alice.

In Message In A Bottle (MIAB), we have designed a protocol where Alice knows the *least possible* about Bob. In fact, we will show that Alice must know Bob's public key, and nothing else. Alice must know at least Bob's public key to authenticate him and be sure she is not talking to a disguised censor. However, contrary to systems like Collage and Telex, there is *no rendezvous point* between Alice and Bob. This may now sound like a needle-in-a-haystack problem: If neither Alice nor Bob know how to contact the other one, how can they ever meet on the Internet? In order to make this possible and reasonably fast, MIAB exploits one of the mechanisms that search engines employ to generate real-time results from blog posts and news articles: *blog pings*. Through these pings, Bob is covertly notified that a new message from Alice is available, and

where to fetch it from. Just like a search engine, Bob can monitor the majority of the blogs published on the entire Internet with limited resources, and in quasi real-time. In some sense, every blog becomes a possible meeting point for Alice and Bob. However, there are over 165 million blogs online, and since a blog can be opened trivially by anybody, for our practical purposes they constitute an infinite resource. We have estimated that, to blacklist all the possible MIAB's drop points, the Censor should block 40 million fully qualified domain names, and four million IP addresses. For comparison, blacklisting a single IP address would block Collage's support for Flickr (the only one implemented), and supporting additional media-hosting sites requires manual intervention for each one.

II. Design

In its essence, MIAB is a system devised to allow Alice, who lives in a country ruled by an oppressive regime, to communicate confidentially with Bob, who resides outside the country. Alice does not need to know any information about Bob except his public key. In particular, MIAB should satisfy these properties: *Confidentiality*, *Availability*, *Deniability*, and *Non-intrusive Deployment*.

To achieve these properties, the MIAB protocol imposes substantial overhead. We do not strive for MIAB's performance to be acceptable for low latency (interactive) communication over the network (such as web surfing). Instead, we want our users to communicate past the Censor by sending small messages (e.g., emails, articles, tweets). The only requirement that Alice must satisfy to use this protocol is to be able to make a blog post. She can create this post on any blog hosted (or self-hosted) outside the Censor's jurisdiction.

The MIAB protocol. Our scene opens with Alice, who lives in a country controlled by the Censor. Alice wants to communicate with Bob, who is residing outside the country, without the Censor ever knowing that this communication took place. To do so, Alice sends a message with the MIAB protocol following these steps:

1) Alice authors a blog post of arbitrary content. The content should be selected to be as innocuous as possible.
2) Alice snaps one or more photos to include in the post.
3) Alice uses the MIAB client software to embed a message M into the photos. The message is hidden using a public-key steganography scheme, using Bob's public key.
4) Alice publishes the blog post, containing the processed photos. Alice can choose the blog arbitrarily, provided it supports blog pings.
5) The blog emits a ping to some ping servers.
6) Meanwhile, Bob is monitoring some of the ping servers, looking for steganographic content encrypted with his public key. Within minutes, he discovers Alice's post, and decrypts the message.
7) Bob reads the message, and acts upon its content (more on this later).

We envision that MIAB might be used to bootstrap more efficient anonymity protocols that require Alice and Bob to know each other a little better (such as Collage, or Telex), but this is outside the scope of this extended abstract.

III. Implementation

To demonstrate the feasibility of MIAB, we have implemented a proof-of-concept application that can be used to post anonymous messages on Twitter, circumventing the block on social networks imposed by Tyria's Censor. To provide a more open evaluation, we have published the code online: The application can be downloaded at http://www.message-in-a-bottle.us, together with our public key. We have set up a small cluster of machines that monitors one of the most popular blog ping servers (webblogs.com), looking for MIAB messages (our cluster plays Bob's part in the algorithm). When a message is found, its content is posted on Twitter under the handle `@corked_bottle`. MIAB relies on Bob's ability to fetch and process all the blog posts created on the Internet in real time. To prove that this is feasible, we have done so with our cluster. Over the period of three months (72 days), we have seen $814,667,299$ blog posts. The average number of posts seen per day is $11,314,823$, and the highest traffic we have experienced is $13,083,878$ posts in a day.

References

[1] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *USENIX Security Symposium*, 2010.

[2] J. Palfrey, H. Roberts, J. York, R. Faris, and E. Zuckerman. 2010 circumvention tool usage report. 2011.

[3] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium*, 2011.

Shellzer: a tool for the dynamic analysis of malicious shellcode

Yanick Fratantonio
Computer Security Lab, UCSB
yanick@cs.ucsb.edu

Christopher Kruegel
Computer Security Lab, UCSB
chris@cs.ucsb.edu

Giovanni Vigna
Computer Security Lab, UCSB
vigna@cs.ucsb.edu

***Abstract*—Shellcode is malicious binary code whose execution is triggered after the exploitation of a vulnerability. The automated analysis of malicious shellcode is a challenging task, since encryption and evasion techniques are often used. This paper introduces *Shellzer*, a novel dynamic shellcode analyzer that generates a complete list of the API functions called by the shellcode, and, in addition, returns the binaries retrieved at run-time by the shellcode. The tool is able to modify on-the-fly the arguments and the return values of certain API functions in order to simulate specific execution contexts and the availability of the external resources needed by the shellcode. This tool has been tested with over 24,000 real-world samples, extracted from both web-based drive-by-download attacks and malicious PDF documents. The results of the analysis show that *Shellzer* is able to successfully analyze 98% of the shellcode samples.**

I. Introduction

Malware, which is a generic term used to denote software that aims to compromise a computer, is the leading threat on the Internet. One of the primary methods used by the attackers to deliver malware is code injection. In the case of web-based malware, the user is lured into visiting a malicious web-page. The JavaScript contained in that page tries to exploit a vulnerability in the browser. If it succeeds, the exploit triggers the execution of an arbitrary piece of code, often called *shellcode*. A shellcode is a small piece of code, whose goal is to compromise the machine that executes it. In this paper, we introduce *Shellzer*, a tool for the dynamic analysis of malicious shellcode. In particular, we focus our attention on shellcode extracted from web-based malware and from malicious PDF documents. Given a shellcode in input, *Shellzer* analyzes it by instrumenting each instruction in the code, to have the complete control over the shellcode's execution. Two different optimizations have also been introduced in order to make this approach feasible in terms of performance. Furthermore, in order to fulfill some specific conditions required for a correct analysis, the tool dynamically alters both the arguments and the return value of some API functions. By doing this, the tool is capable of observing the behavior of the shellcode during a real-attack scenario. As output, the tool returns a report that contains the following information: a *complete* trace of the API calls (with their most significant arguments and their return values), the URLs from which external resources have been retrieved, and the evasion techniques used by the shellcode. Furthermore, the tool returns the additional binaries that are retrieved at run-time, even if the binary was originally encrypted (note how just having the encrypted binary would be useless, since the decryption routine is implemented in the shellcode, and not in the binary itself). This feature is useful also when dealing with shellcode samples extracted from malicious PDF documents, where the additional payload is contained in the PDF document itself, usually encrypted.

II. Overview of the system

Shellzer dynamically analyzes shellcode samples by instrumenting their execution at a single-instruction level granularity. The instrumentation is performed by using PyDbg [1], a Python Win32 debugging abstraction class. In particular, the core of the analysis is performed in the `EXCEPTION_SINGLE_STEP`'s handler that it is called between the execution of each assembly instruction. Our goal was to have complete control over the shellcode's execution, as if we were using an approach based on emulation. The advantage in using such a technique is that we can dynamically decide if it is necessary to single-step through the code or not, so that the overhead caused by the instrumentation is introduced only when it is strictly required. We will now discuss the three main components of our system, the motivations behind our design decisions and the challenges that each component is addressing.

API calls detection and tracing. We now describe how *Shellzer* detects and traces the API functions called by the shellcode. Before the execution of each assembly instruction we retrieve the program counter (`PC`), and we then determine where the `PC` is pointing to. By checking if the `PC` is pointing to a memory region where a specific Windows API function is located, we are able to detect that an API is called, and which one it is. Moreover, by retrieving the stack pointer (`SP`), we are also able to determine the values of the API's arguments, that have to be placed on top of the stack just before the API is called. By using a similar technique, we also retrieve the API's return value. Note that since we have the complete control over the shellcode's execution, this technique is powerful enough to detect and properly handle a number of assembly-level evasion techniques that real-world shellcode we found in our dataset often use.

Dynamic interaction. By using an approach based on single-step instrumentation, we are able to inject custom pieces of code at any moment between the execution of each assembly instruction, and, therefore, the tool has the ability

Listing 1. calc_exit_points()

```
def calc_exit_points(loop_body, known_loops):
  exit_points = set()
  candidates = set()
  for address in loop_body:
    if address in known_loops.keys():
      candidates.add(known_loops[address])
      continue
    ins = disassemble(address)
    if ins in branches:
      taken, not_taken = get_dest_addrs(ins)
      candidates.add(taken)
      if not_taken not None:
        candidates.add(not_taken)
  for candidate in candidates:
    if candidate not in loop_body:
      exit_points.add(candidate)
  known_loops[loop_body[0]] = exit_points
  return exit_points
```

to dynamically read and write the process memory, access and modify the CPU registers' values, and so on. Our tool exploits this capability to address two of the big challenges we found in analyzing shellcode samples. The first issue comes from the fact that the shellcode might try to retrieve additional resources (usually additional malware) and, if they are not available, the shellcode's execution might crash. This outcome is not desirable since our goal is to analyze the behavior of the shellcode as if it were executed during a real-world attack. Therefore, if the shellcode fails to retrieve these external resources, we dynamically simulate that those resources are available, by properly altering the return values of some specific Windows API functions. The second issue we addressed is related to the fact that some shellcode need to be executed in a very specific execution context. In order to solve this problem, we simulate that the whole analysis is performed within the required execution context. Also in this case, this is obtained by modifying on-the-fly the content of some specific memory regions and the return values of specific Windows API functions.

Performance speed-up. Instrumenting the whole shellcode's execution with a single-instruction granularity gives us the complete control over the shellcode's execution, but bears a significant performance overhead. For this reason, we implemented two optimizations that allow for the disabling of the single-step mode when it is not necessary. The first aims to disable the single-step mode during the execution of API functions, whose semantic is already known. From the technical point of view, this is achieved by performing the following steps: once we detect that an API is about to be called, we determine the return address (the address in memory where the execution will return to after the API's execution), we set a software breakpoint on it, and we continue the execution of the API with the single-step mode disabled. When the execution will return from the API's code, the breakpoint will be triggered and, after having restored the single-step mode, we continue with the normal analysis. The second optimization is related to the following observation: despite the fact that shellcode is usually few hundreds of bytes long, the number of instructions that are actually executed at run-time is in the order of millions due to the presence of many loops. What we noticed is that once the loop's body has been analyzed the first time, the single-step instrumentation is often no longer needed for the other iterations. To handle this we designed an algorithm to temporarily disable the single-step mode during the execution of loops. We now discuss our algorithm. After detecting that the execution is in loop (to do that, the tool keeps track of the instructions that have been previously executed), we analyze the loop's body (that is the list of instructions that constitute the loop itself) and we determine which are the exit points (i.e., the set of addresses such that at least one of them has to be reached once the loop's execution is terminated). To properly handle nested loops, we needed to store the information related to the previously determined loops. The complete algorithm is presented in Listing 1. Once the set of exit points is determined, we set an hardware breakpoint on each of them and, after disabling the single-step mode, the execution is resumed. When the loop's execution terminates, one of the breakpoints will eventually be triggered and we will have a chance to resume the single-step mode. The impact of this optimization is tremendous: in fact, just few hundreds of instructions have to be single-step executed, while the number of instructions actually executed is in the order of millions, and the analysis consequently takes few seconds instead of several minutes.

III. Evaluation

We evaluated *Shellzer* by analyzing 24,214 real-world shellcode samples, previously detected and extracted by Wepawet [2], an online service for detecting and analyzing web-based malware, malicious PDF documents and others. *Shellzer* has been able to analyze most of them, and the analysis didn't finish for our tool's limitations only in the $\sim 2\%$ of the cases.

IV. Conclusion

In this paper, we have presented *Shellzer*, a tool for the dynamic analysis of shellcode samples. Thanks to a series of optimizations, the single-step instrumentation turned out to be a successful approach. Starting from November 2011, *Shellzer* is used by Wepawet to process the shellcode samples detected during its analysis. Over 148,000 samples have been successfully analyzed since then, and useful insights into real-world malware have been provided to security researchers from around the world.

References

[1] P. Amini, Pydbg, http://pedram.redhive.com/PyDbg/
[2] M. Cova, Wepawet, http://wepawet.cs.ucsb.edu

User-Perspective Augmented Reality Magic Lens

Domagoj Baričević, Cha Lee, Matthew Turk, Tobias Höllerer
Four Eyes Lab
University of California, Santa Barbara
{domagoj, chalee21, mturk, holl}@cs.ucsb.edu

(a) (b) (c)

Figure 1: Prototype of a user-perspective AR magic lens. (a) shows the hardware used: Kinect, Wiimote, and display. The Kinect sensor is mounted behind the tablet facing out and the Wiimote is mounted above the display facing the user. (b) shows a device-perspective magic lens. The image on the magic lens is from the perspective of the Kinect sensor. (c) shows the user-perspective magic lens. The image on the magic lens is from the perspective of the user (the small misalignment is due to imperfect calibration).

1 Introduction

One of the most common interface types in Augmented Reality (AR) is the metaphor of the hand-held magic lens [1]. In this type of interface a mobile device such as a phone or tablet is used as a window that can reveal virtual content aligned with the real world. Concept images of AR magic lenses show that the magic lens displays a scene from the user's perspective, as if the display were a smart transparent frame allowing for perspective-correct overlays. This is arguably the most intuitive view. However, current AR magic lens implementations rely on video-see-through methods and show the augmented scene from the point of view of the camera on the hand-held device. The perspective of that camera is very different from the perspective of the user (compare Figure 1b to Figure 1c), so what the user sees does not align with the real world. A true magic lens would show the scene from the point of view of the user, not the device. The technology needed to make this possible has only recently become available. In this paper we present the first proof-of-concept prototype of a hand-held AR magic lens with user-perspective rendering.

2 Implementation

We developed a prototype implementation of a tablet-sized user-perspective AR magic lens. The prototype has many limitations compared to an ideal AR magic lens, but it serves as a fully functional proof of concept. It was built using off-the-shelf hardware and open source software.

There are three challenges in presenting user-perspective imagery: (1) tracking the user's head position accurately, (2) tracking the pose of the display with respect to the world accurately, and (3) obtaining/creating a model of the world. Generating user-perspective images requires all three pieces of information since we must render the world from an arbitrary observer viewpoint different from the device's viewpoint.

2.1 Hardware

The prototype was built using three off-the-shelf devices and a desktop workstation. The Kinect sensor is the central component of the prototype and was mounted behind the hand-held display, as seen in Figure 1. The display has a Lilliput 10.4" TFT LCD screen with a native resolution of 800x600 pixels. Attached to the top of the display is a Wiimote which is used to track the user's head position. The user wears a pair of goggles with four infra-red LED markers attached. The Wiimote communicates with the workstation via Bluetooth while both the Kinect and the display are tethered to the workstation. The workstation runs Kubuntu and has 32GB of RAM, two dual-core AMD Opteron 2.60GHz CPUs, and two NVIDIA GPUs: Quadro FX 5600 and Quadro 6000.

2.2 Software

In order to provide a user-perspective view of the augmented scene, three requirements need to be met: a means of tracking the user, a way to reconstruct the real-world scene, and a way to track the display with respect to the scene. Our system uses two separate software components to perform these tasks. The main component performs the reconstruction of the scene and tracks the display at the same time. It also handles the rendering of the user-perspective image of the AR scene. A separate standalone component performs the user tracking using the Wiimote and four LED markers on a

pair of goggles worn by the user. Communication between the two is achieved by using a UDP network connection over a loop back interface; this makes the overall system more modular, with easy support for alternate tracking solutions.

The main software component deals primarily with the display and the Kinect sensor. It generates and uses two models of the real world scene. The first is the live feed from the Kinect sensor that is represented as a point cloud. The second is a reconstructed model of the real world using a modified version of KinFu, the Point Cloud Library's implementation of the Kinect Fusion algorithm [2]. Our modifications were to the ray-caster which renders the reconstructed model, adding support for rendering the reconstructed volume both in color and from arbitrary viewpoints. KinFu is also used to track the pose of the Kinect relative to the real-world scene. The algorithm automatically computes the pose of the Kinect relative to KinFu's internal coordinate system and this is used when positioning augmentations that are defined in terms of a real-world coordinate system. This requires an initialization step where the magic lens is aimed at a marker defining the coordinate system and the pose of the Kinect is computed relative to this marker.

The final AR scene is composed by blending three graphical layers. The base layer is the colored rendering from the modified KinFu raycaster. The mid layer is a reprojection of the point cloud from the live depth feed. The final layer has the augmentations to the scene, rendered with proper occlusion with the real world. Proper occlusion is accomplished by writing the depth map from KinFu into the top layer's depth buffer before the augmentations are rendered.

2.3 Performance

The prototype has two major constraints inherited from the Kinect sensor: it can only detect depth in the range of approximately 0.5 to 4 meters, and cannot be used in spaces with strong IR illumination such as outdoor environments. The depth range constraint effectively prevents any practical application requiring users to manipulate objects with one hand, while holding the tablet in the other. However this is a current limitation of the technology and could be removed in the future as the technology improves and more depth cameras can be used to cover a wider depth range and space.

The KinFu algorithm also imposes additional constraints. It automatically defines a portion of the space that it will reconstruct and anything outside this space will not be reconstructed and the depth data will be ignored. Although the size of the volume is arbitrary, it is always modeled as a 512x512x512 voxel space. This results in a trade-off between the size of the volume that can be reconstructed and the level of detail. The colored rendering of the voxel space in particular has a rather voxelated appearance, due to the large amounts of pixel data that are lost when building the 3D color model. The coarseness of the reconstruction also results in small errors with the occlusions between real and virtual objects, particularly along the edges of the objects. These drawbacks are partially compensated by also using the point cloud from the live feed. This point cloud is not constrained to a given area of the real world and offers much better color representation as it uses all the current color data from the Kinect sensor. The downside of the point cloud is that it only uses the currently available depth data which generally has gaps and is constrained to what the Kinect sees, not what the user sees.

Combining both the reconstruction from KinFu and the live point cloud from the Kinect sensor results in a representation of the real-world space that is of greater quality that either can offer alone. Unfortunately, the KinFu algorithm will sometimes wrongly estimate the pose of the Kinect, causing a misalignment between the two models. Also, if the pose estimation fails completely, the algorithm will reset the reconstruction and it will need to be reinitialized.

3 Conclusion

We have implemented a user-perspective AR magic lens, and to our knowledge this is the first one of its kind. Our prototype was implemented using current hardware and software with minor modifications. As seen in Figure 1, it still produced visibly acceptable results. Unlike the common AR magic lens of the device-perspective type, the prototype must reconstruct the world in real-time and render it from a different view. The depth range, limited image quality, and tracking robustness does not allow for any truly practical applications yet with this device. But this prototype does show that it may soon be possible. Better and more numerous depth sensors will reduce the depth range issue while increasing the resolution of the 3D model. We can also expect that handheld devices will soon have the processing power needed to make an untethered version of this prototype possible. Overall, given the complexity of capturing and re-rendering user-perspective arbitrary scenes in real-time, the results we achieved with off-the-shelf hardware and software are very encouraging.

Other methods could be used to create a user-perspective magic lens. One approach would be to use a transparent screen for optical-see-through AR; this would introduce issues such as binocular rivalry and visual interference as users try to align virtual objects rendered on a transparent layer against objects in the background. However, adding a parallax barrier for an autostereo transparent display would greatly reduce this issue. In fact, this would be a promising solution as no reconstruction would be necessary so there would be no visual artifacts or errors with the real world content.

References

[1] E. Costanza, A. Kunz, and M. Fjeld. Mixed reality: A survey. In *Human Machine Interaction*, volume 5440 of *Lecture Notes in Computer Science*, pages 47–68. Springer Berlin / Heidelberg, 2009.

[2] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

Sketch Recognition (Practically) Anywhere with SPARK

Jeffrey Browne and Timothy Sherwood
University of California, Santa Barbara
{jbrowne, sherwood}@cs.ucsb.edu

***Abstract*—The sketch community has, over the past years, developed a powerful arsenal of recognition capabilities and interaction methods. Unfortunately, many people who could benefit from these systems lack pen capture hardware and can only draw diagrams on traditional surfaces like paper.**

In this work we explore bringing the benefits of sketch capture and recognition to traditional surfaces through a common smartphone with SPARK, a framework for building mobile, image-based sketch recognition applications. As evidence of our techniques, we have implemented a mobile app in SPARK that captures images of Turing machine diagrams drawn on paper, a whiteboard, or even a chalkboard, and allows users to simulate recognized Turing machines on their phones.

I. Introduction

Sketch recognition applications have, over the past several years, managed to bring the power of sophisticated design applications within reach of novices by leveraging users' previous freehand drawing skills. However, the hardware required to capture these drawn strokes remains niche, and the reality is that most design sketching today still happens on traditional whiteboards, paper, and even chalkboards.

In order to enable sketch recognition for everyday users, we have implemented the *Sketch Practically Anywhere Recognition Kit* (SPARK), a mobile, image-based framework for developing sketch recognition apps on ubiquitous smartphone hardware, and as a proof of concept of the framework, we have developed a Turing machine simulator within the framework (figure 1). The result is a full system that allows users to capture images of Turing machine diagrams drawn on paper, whiteboards, or even chalkboards, and to simulate them step-by-step on a mobile device.

Static image data contains no temporal information to determine how and when strokes are drawn, but through novel modifications to known image processing techniques, our framework can reliably capture hand-drawn marks and convert them to a form more readily handled by many sketch processing systems—"strokes" consisting of a totally-ordered path of X,Y coordinate pairs.

II. User Experience

To interact with our system, an end user first draws a Turing machine on a whiteboard. When she wishes to analyze her work, she photographs the relevant diagram, and a picture is sent to a recognition server.

In a matter of seconds, the server returns the recognized Turing machine recognized from the photo, which is displayed in "Simulation" mode on the device (figure 1), and the user can begin stepping through the operation of the Turing machine straight away. The I/O tape displayed below the diagram updates upon each press of the step button, with the states gaining or losing highlight depending on their *active* status. Through a menu system, she can define a different tape string to be placed on the tape for simulation, or save the recognized Turing machine for later examination.

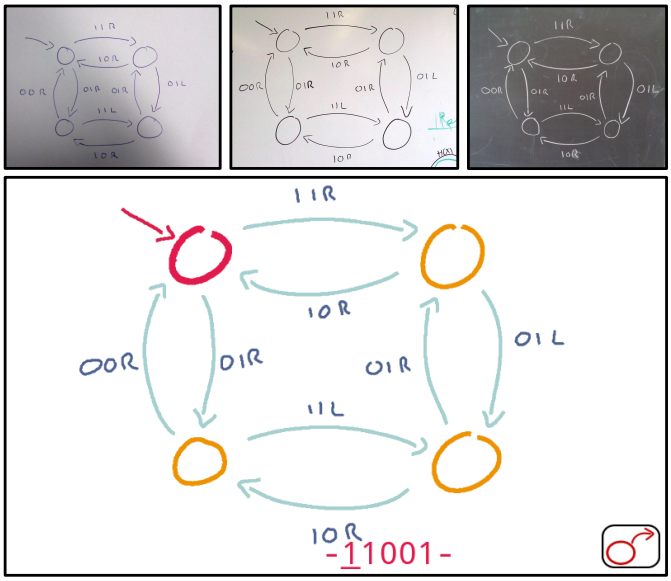


Fig. 1: Apps within SPARK (like this Turing machine simulator) use sketch recognition techniques on strokes extracted from photos of whiteboards, paper, and chalkboards.

III. Recognizing Strokes from an Image

SPARK's architecture is such that, after a user takes a photograph of her diagram, the stroke extraction and recognition occurs on a remote server, a process that can be broken down into three distinct modules:

A. Image Normalization

Since we aim to recognize marks on various surface types, the first step in image normalization determines if the diagram is drawn on a whiteboard/paper, or a blackboard. Our system exploits the tail of an image's value histogram, where values correspond to the ink, rather than the median pixel value, which corresponds to the color of the background surface,

since neither board type is distinctly black or white, but rather more of a gray. If there are more pixels significantly lighter than the median than pixels that are darker, we assume the ink is light (e.g. chalk on a blackboard), and we invert the image, so our algorithms can proceed assuming dark ink.

Further, when a user takes a photograph of her diagram, shadows, reflections, and smudges from unclean boards result in an image where static threshold values for ink and background separation will not suffice. To separate foreground marks from the background surface, our algorithm performs a modification of Niblack's adaptive thresholding[**?**]; we generate a signature image of the image background to subtract out of the original image, which is then binarized according to a static threshold.

B. Stroke Extraction

it is represented as regions of solid black against a white background, with potentially dozens of pixels corresponding to each "point" in the logical stroke that they represent.

After image normalization, the marks can be separated from the board, but the bitmap regions representing ink must be further transformed into a logical path of points. First, to determine the (unordered) set of points corresponding to each symbol, we utilize established thinning techniques[**?**], modified to take into account the local stroke thickness at each point to compensate for errors that result from the limited context inherent in the 3x3 sliding window-based algorithm. For example, a "spurious edge" error occurs when a pixel mistakenly forms what looks like a legitimate endpoint early in the thinning process, resulting in a hair-like protrusion from the logical center-line of a stroke.

To trim these edges, our graph building algorithm first selects intersection points between simple edges as part of the graph of stroke points, finding the original ink's thickness around that point. It then iteratively merges the closest point that falls outside the graph's total thickness region until all thinned points either in the graph or covered by another point. Spurious edges fall entirely into the thickness region of another point and are omitted from the final point graph.

Since thinning erodes regions to a medial axis, regions where strokes cross can be split into multiple distinct intersections in the final graph. To address this shortcoming, our modified algorithm examines a circular region with a radius of the stroke's thickness around every intersection in the adjacency graph, where the thinning results are reexamined. If multiple intersections overlap according to their thickness regions, then the algorithm creates a point with their average coordinates to use as a single replacement point.

Once the points of each mark are found, our algorithm then imposes an ordering on them, effectively "tracing" one or more strokes per thinned ink region by traversing the point graph starting at an endpoint (or arbitrary point if no endpoint exists). If, during traversal, our algorithm encounters an intersection of multiple line segments, the path is extended preferring the overall smoother path. Tracing is complete when all points in the graph are part of a stroke, at which point the entire collection of strokes is passed to the recognition phase.

C. Recognition over Extracted Strokes

Recognition for apps within SPARK begins when, one at a time, the strokes are submitted to a hierarchical recognition module. Individual letters are recognized using a feature-based linear classifier, which, as with heuristic logic for recognizing closed shapes, and arrows, analyzes raw stroke data. When one of these "low-level observers" recognizes its target pattern, it builds a semantic description of the symbol in the form of an *annotation* that is then tagged onto the associated set of strokes. While lower-level recognizers examine stroke data directly, text string-, directed graphs-, and Turing machine recognizers are implemented as "higher-level observers" that respond only to the addition of semantic annotations, and as such they mostly analyze the structure among recognized symbols. Specifically, text strings are recognized from individual letters, assuming they are of a similar scale and proximity and share a common baseline, whereas directed graphs are collected from closed shapes and arrows annotations depending on arrow direction and proximity to nodes. Finally, each Turing machine is built from text strings with length three as labels and directed graphs as the machine's logic.

After all of the strokes extracted from the image along with their annotations have propagated through the recognition process, semantic meaning of the diagram is sent in XML format to the mobile phone. From here, a user can interact with her diagram using a mobile interface as already discussed.

IV. Discussion

An important consideration for any sketch recognition application is the rectification of inevitable (but hopefully infrequent) errors. Sketch is inherently ambiguous as an input mode, and without user guidance, some drawings would be recognized incorrectly, even by human observers. Currently, the only means for correction are modifying the original drawing and re-capturing an image, but we foresee a mode where users will select strokes using touch, on which they will impose the desired interpretation manually.

References

[1] W. Niblack, *An introduction to digital image processing*. Prentice-Hall International, 1986.

[2] N. J. Naccache and R. Shinghal, "An investigation into the skeletonization approach of hilditch," *Pattern Recognition*, vol. 17, no. 3, pp. 279–284, Jan. 1984.

Catch Me If You Can: Pursuit and Capture in Polygonal Environments with Obstacles

Kyle Klein and Subhash Suri

Department of Computer Science
University of California
Santa Barbara, CA 93106
{kyleklein, suri}@cs.ucsb.edu

***Abstract*—We resolve a several-years old open question in visibility-based pursuit evasion: how many pursuers are needed to capture an evader in an arbitrary polygonal environment with obstacles? The evader is assumed to be adversarial, moves with the same maximum speed as pursuers, and is "sensed" by a pursuer only when it lies in line-of-sight of that pursuer. The players move in discrete time steps, and the capture occurs when a pursuer reaches the position of the evader on its move. Our main result is that $O(\sqrt{h} + \log n)$ pursuers can always win the game with a deterministic search strategy in any polygon with n vertices and h polygonal obstacles (holes). In order to achieve this bound, however, we argue that the environment must satisfy a *minimum feature size* property, which essentially requires the minimum distance between any two vertices to be of the same order as the speed of the players. Without the minimum feature size assumption, we show that $\Omega(\sqrt{n})$ pursuers are needed in the worst-case even for *simply-connected* (hole-free) polygons of n vertices!**

I. Introduction

Pursuit evasion games arise in applications ranging from military strategy to trajectory tracking, search-and-rescue, monitoring, surveillance and so on [1]. The general problem addressed in this paper relates to pursuit evasion in *continuous* space under a visibility-based model of sensing. In particular, we have an environment modeled as a polygon P, possibly containing obstacles. We use h to denote the number of obstacles, a mnemonic for "holes" in the polygon, and n to denote the total number of vertices in the environment, including the obstacles. A team of pursuers $\{p_1, p_2, \ldots, p_k\}$ is tasked to locate and catch a mobile evader e, and the fundamental question we seek to answer is this: *how many pursuers are always sufficient to catch the evader, no matter what strategy it follows?* Specifically, as a function of n and h, what is the smallest number k of pursuers that guarantees a win for the pursuers in any polygon of n vertices and h holes.

The most relevant work to our research is the paper by Guibas et al. [2], which introduced a formal framework and analysis of visibility-based pursuit in complex polygonal environments. In order to make the problem tractable, however, Guibas et al. make one crucial simplifying assumption: *the evader loses if it is "seen" by any pursuer.* That is, the pursuers need to only detect the presence of the evader, and not physically catch it. With this weaker requirement of "capture," Guibas et al. manage to prove several interesting combinatorial bounds, including that $\Theta(\log n)$ pursuers in a simply-connected polygon, and $\Theta(\sqrt{h} + \log n)$ pursuers in a polygon with h holes (obstacles), are always sufficient and sometimes necessary. In fact, these bounds hold even if the evader can move arbitrarily faster than the pursuers.

In the intervening twelve or so years, there has been little progress on extending these "detection" of evader bounds to physical "capture" of the evader. In fact, there are only a handful of small results to speak of. First, Isler et al. [3] show that in simply-connected polygons, two pursuers can capture the evader in *expected* polynomial time using a randomized strategy. Recently, and almost simultaneously, two groups, [4] and [5], proved that *if the location of the evader is always known to the pursuers*, e.g., using an ubiquitous camera network, then 3 pursuers are enough to win the game. Without these extreme conditions of unfair advantage to the pursuers, no non-trivial upper bound on the number of pursuers necessary to win the game is known. The main result of this paper is to resolve this question.

Our first result is a general lower bound of $\Omega(\sqrt{n})$ for the number of pursuers needed in the worst-case to catch an equally fast evader in *simply-connected* (hole-free) polygons of n vertices. Next we show that if the environment satisfies a *minimum feature size* property, then $O(\sqrt{h} + \log n)$ pursuers are always sufficient to catch the evader in a polygon with n vertices and h obstacles (holes). When the polygon is simply-connected (hole-free), this yields an $O(\log n)$ bound for the number of pursuers. Due to space constraints we omitt formal proofs, however, we sketch the general ideas where possible.

II. Problem Formulation

The pursuers $\{p_1, p_2, \ldots, p_k\}$ and the evader e take alternate turns, where all the pursuers can move simultaneously on their turn. In each turn, a player can move to any position that is within distance one of its current location, under the shortest path metric, in the *free space* (the region of the polygonal environment not occupied by holes). The players' sensing model is based on line-of-sight visibility: a pursuer p sees the evader e only if the line segment (p, e) does not intersect the boundary of P. The pursuers are assumed to operate in a global communication model, allowing them to coordinate their moves, and share knowledge of the evader's location.

We assume that all the players know the environment, and the pursuers do not know the strategies or the future moves of the evader, although the evader may have complete information about pursuers' strategies. The pursuers *win* if after a finite amount of time, some pursuer becomes collocated with the evader. The evader wins if it can evade indefinitely.

III. Upper and Lower Bounds for Capture

Without any restrictions on the environment or the speed of the players, except that pursuers and evader have the same maximum speed, we can show the following lower bound.

Theorem 1. *There exist simply-connected (hole-free) polygons with n vertices that require at least $\Omega(\sqrt{n})$ pursuers to catch an equally fast evader.*

We now describe a simple geometric condition on the environment, which we call the *minimum feature size* condition, which allows us to circumvent the lower bound of Theorem 1. The minimum feature size is defined as follows.

Definition 1. Minimum Feature Size: *The minimum feature size of a (multiply-connected) polygon P is the minimum distance between any two vertices, where the distance is measured by the shortest path within the polygon.*

We will assume that the minimum feature size of the environment is *lower bounded* by the maximum speed of the players: i.e., the environment has minimum feature size of at least one. We feel that this condition naturally occurs in physical systems, and its violation is the source of the $\Omega(\sqrt{n})$ lower bound. As a result, a formal proof establishes the following bound for capture in hole-free polygons.

Theorem 2. *Suppose P is a simply-connected polygon of n vertices satisfying the minimum feature size, then $O(\log n)$ pursuers can always capture the evader.*

Next we give a high-level description of the pursuers' winning strategy in the presence of holes. The search is based on a divide-and-conquer strategy that recursively partitions the environment, installing some pursuers to guard the "separating triangles," until the search reaches simply-connected regions, at which point we can invoke Theorem 2. In particular, the strategy has the following main steps.

Algorithm RecursivePursuit

1) Identify $O(\sqrt{h})$ triangles that partition P into two sub-polygons, each containing at most $2h/3$ holes.
2) Guard each "separating triangle" with a constant number of pursuers so that the evader cannot move across the partition without being captured.
3) Recursively search one side of partition, then the other.
4) The recursion stops when the sub-polygon has no holes. Then, use Theorem 2 to search for, and if present, capture the evader.

The algorithm begins by triangulating the environment P. Recall that the graph-theoretic *dual* of the triangulation is a planar graph, with a vertex for each triangle and an edge between two nodes if those triangles have a common boundary edge. We reduce the size of this graph by *contracting* each vertex of degree 2, and deleting every vertex of degree one. (See Figure 1 for an illustration.) In the end, each surviving vertex has degree 3, and Euler's formula for planar graph implies that G has h faces, at most $2h-2$ vertices, and at most $3h-3$ edges. We then use the well-known Planar Separator Theorem [6] to partition G into two parts, each containing at most $2/3$ the nodes of G, by deleting a set of $O(\sqrt{h})$ nodes. In the geometric space, this separator corresponds to a set of $O(\sqrt{h})$ triangles that divide the environment P into two parts, each containing at most $2h/3$ holes. (See Figure 1(d).)

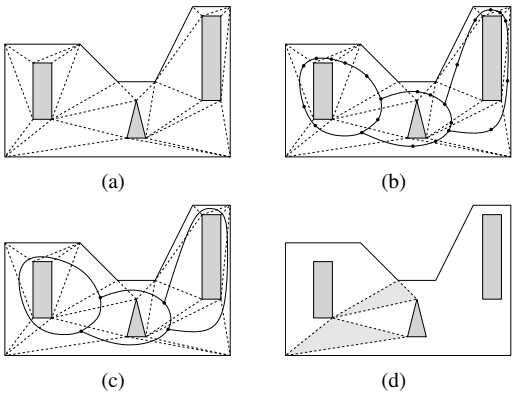


Fig. 1. (a) A triangulation; (b) its dual graph; (c) the contracted dual; (d) illustration of the recursive strategy—two separating triangles (shaded) break the environment into 3 connected regions, each with at most 1 obstacle.

The remaining difficulty, is to show how to guard each of the separating triangles with a constant number of pursuers to prevent the evader from escaping across the partition. While we omitt a discussion of how this is accomplished, we note that this is only possible due to the minimum feature size restriction on the environment.

With those pieces in place, one can show that the number of pursuers needed is $O(\sqrt{h}+\log n)$, as follows. The $O(\log n)$ pursuers used in the base case are *reused* throughout the divide and conquer strategy, so they form only an additive term. The remaining demand for the pursuers are those placed at the separating triangles throughout the divide-and-conquer. Their count has the following recurrence: $T(h) = T(2h/3)+c\sqrt{h}$, where c is a constant. This well-known recurrence solves to $T(h) = O(\sqrt{h})$, easily proved by induction, and so the total number of pursuers used by the algorithm is $O(\sqrt{h}+\log n)$, and we state our main Theorem.

Theorem 3. *Let P be a polygonal environment with n vertices and h disjoint holes satisfying the minimum feature size. Then, $O(\sqrt{h}+\log n)$ pursuers can always capture the evader.*

IV. Conclusion

We have shown that $O(\sqrt{h}+\log n)$ pursuers can always capture an equally fast evader in any polygon with n vertices and h obstacles (holes). This matches the best bound known for just *detecting* an evader (a simpler problem) in the visibility-based pursuit. Further, we show that capturing the evader is *provably* harder than detecting, because capturing without the minimum feature size requires $\Omega(\sqrt{n})$ pursuers (in simply-connected polygons) while detection can be achieved with just $O(\log n)$ pursuers, even against an arbitrarily fast evader [2].

References

[1] T. H. Chung, G. A. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Auto. Robots*, pp. 1–18, 2011.
[2] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "Visibility-based pursuit-evasion in a polygonal environment," *IJCGA*, vol. 9, no. 5, pp. 471–494, 1999.
[3] V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion in a polygonal environment," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 875 – 884, 2005.
[4] D. Bhadauria and V. Isler, "Capturing an evader in a polygonal environment with obstacles," in *Proc. of the Joint Conf. on Artificial Intelligence (IJCAI)*, 2011, pp. 2054–2059.
[5] K. Klein and S. Suri, "Complete information pursuit evasion in polygonal environments," in *Proc. of 25th Conference on Artificial Intelligence (AAAI)*, 2011, pp. 1120–1125.
[6] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem," oct. 1977, pp. 162 –170.

Quantum Algorithms for the Hidden Subgroup Problem for Semi-Direct Product Groups

Wim van Dam
Departments of Computer Science and Physics,
University of California Santa Barbara,
Santa Barbara, CA 93106, USA
Email: vandam@cs.ucsb.edu

Siladitya Dey
Department of Computer Science,
University of California Santa Barbara,
Santa Barbara, CA 93106, USA
Email: siladitya_dey@cs.ucsb.edu

***Abstract*—The Hidden Subgroup Problem is a framework that has admitted quantum algorithms to perform exponentially faster than their classical counterparts. However the problem in the generic case of non-abelian groups presents no known efficient quantum algorithm. In this article, which is a work in progress, we introduce a novel approach at formulating an efficient quantum algorithm in the specific case of $\mathbb{Z}/p^r\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z}$ groups.**

I. Introduction

The quantum algorithm to factorize integers as given by Shor [1] is exponentially faster than any known classical method. This led to increased research in the area of quantum computing, resulting in more quantum algorithms which were exponentially faster than their classical counterparts (discrete logarithm problem [1], Generalized Simon's problem [2]). In 1995, Kitaev [3] showed that the above algorithms and more could be generalized to the problem of finding subgroup generators of a group using evaluations of a function that "hides" the subgroup [8]. This generalized framework is the Hidden Subgroup Problem (referred hereon as HSP) and has been successful in admitting quantum algorithms which are exponentially faster than their classical counterparts. Moreover, it is known that there exists an efficient solution to the HSP for generic abelian groups, but not for non-abelian groups. Motivation for research in this area stems from the knowledge that an efficient solution to the HSP over the symmetric group (dihedral group) will result in an efficient quantum algorithm for graph isomorphism (shortest vector in a lattice). In this article, we consider a specific class of non-abelian groups, i.e. the semi-direct product groups.

II. Related Work

There has been considerable work in trying to solve the HSP in semi-direct product groups and in this article we attempt to extend it to the generic case of $\mathbb{Z}/p^r\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z}$. Previous work includes reducing the HSP in the dihedral group to finding cyclic subgroups of order 2, [4]. Later, it was shown that the HSP in $\mathbb{Z}/N\mathbb{Z} \rtimes \mathbb{Z}/q\mathbb{Z}$, for positive integers N, q such that $N/q = \mathrm{poly}(\log N)$, reduces to finding cyclic subgroups of order q and this can be solved efficiently [5]. In 2007, an efficient HSP algorithm in $(\mathbb{Z}/p^r\mathbb{Z})^m \rtimes \mathbb{Z}/p\mathbb{Z}$, with p prime and integers r, m was found [6] Following this, in 2009 an efficient quantum algorithm to solve the HSP in $\mathbb{Z}/p\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z}$ for distinct odd primes and $s > 0$ such that $p/q = poly(\log p)$ was shown [7].

III. The Semi-direct Product Group $G := \mathbb{Z}/p^r\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z}$

In this section, we describe the properties of the group G in detail. The semi-direct group is defined as below,

$$G := \mathbb{Z}/p^r\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z} \tag{1}$$

with p, q prime and $r, s \in \mathbb{Z}^+$. Let $\mathbb{Z}/p^r\mathbb{Z}$ and $\mathbb{Z}/q^s\mathbb{Z}$ be finite, cyclic, abelian groups and $\phi : \mathbb{Z}/q^s\mathbb{Z} \to Aut(\mathbb{Z}/p^r\mathbb{Z})$ be the homomorphism that defines G, with $\phi(0)$ the identity operation on $\mathbb{Z}/p^r\mathbb{Z}$. The elements of G are (a, b) where $a \in \mathbb{Z}/p^r\mathbb{Z}, b \in \mathbb{Z}/q^s\mathbb{Z}$. The product of the group elements are defined as, $(a, b)(c, d) = (a + \phi(b)(c), b + d)$, the inverse as, $(a, b)^{-1} = (\phi(-b)(-a), -b)$ and $(0, 0)$ is the identity of G. The order of the group is $p^r q^s$ and by *Burnside's Theorem* (*Theorem* 9.3.2, [9]) we know that G is *solvable*.

We note that G has only two Sylow subgroups (up to isomorphism), the Sylow p-subgroup $\mathbb{Z}/p^r\mathbb{Z}$ and the Sylow q-subgroup $\mathbb{Z}/q^s\mathbb{Z}$ both of which are also cyclic. This enables us to employ an existing theorem (*Theorem* 9.4.3, [9]) which states that G is *metacyclic*. Moreover by definition of G, $N := \mathbb{Z}/p^r\mathbb{Z} \times \{0\}$ is a normal subgroup of G, and the quotient group is $G/N = \{xN|x \in G\} = \{(0,h)N|h \in \mathbb{Z}/q^s\mathbb{Z}\}$. Since $(0,h_1)N(0,h_2)N = (0,h_1)(0,h_2)NN = (0,h_1+h_2)N$. We can now see that G admits a finite normal series as, $(A_0 =)G \lhd (A_1 =)\mathbb{Z}/p^r\mathbb{Z} \lhd (A_3 =)(0,0)$, and hence G is *supersolvable*. This means that every subgroup, $H \leq G$ is also supersolvable.

IV. Irreps of G

In this section, we give important properties of the irreducible representations (hereon known as *irreps*) of the group G. The motivation to use representation theory was natural due to its formulation; group elements are matrices and the group operation is represented as matrix multiplication. Moreover, the check for reversibility of a quantum algorithm is akin to checking for unitarity of the operations involved, which is straightforward in linear algebra.

The *little group method* of Wigner and Mackey can be used to enumerate the irreps of $G = \mathbb{Z}/p^r\mathbb{Z} \rtimes \mathbb{Z}/q^s\mathbb{Z}$ [10] . We designate $G = A \rtimes H$ such that $A = \mathbb{Z}/p^r\mathbb{Z}$ and $H = \mathbb{Z}/q^s\mathbb{Z}$, and note that A is abelian. Using the above-mentioned method we enumerate the irreps of the following dihedral group, $G = \mathbb{Z}/3\mathbb{Z} \rtimes \mathbb{Z}/2\mathbb{Z}$ as having two irreps of dimension 1 given by, $\forall g \in G, \Pi(g) = 1$, and

$$\Pi(g) = \Pi((a,h)) = \begin{cases} 1 & \text{if } h = 0 \\ -1, & \text{if } h = 1 \end{cases}$$

$$\Pi(g) = \begin{cases} 1 & \text{if } g \in A \\ -1, & \text{otherwise} \end{cases}$$

and one irrep of dimension 2 given by,

$$\Pi((0,0)) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Pi((1,0)) = \begin{bmatrix} \exp(\frac{i2\pi}{3}) & 0 \\ 0 & \exp(\frac{i4\pi}{3}) \end{bmatrix},$$

$$\Pi((2,0)) = \begin{bmatrix} \exp(\frac{i4\pi}{3}) & 0 \\ 0 & \exp(\frac{i2\pi}{3}) \end{bmatrix}, \Pi((0,1)) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

$$\Pi((1,1)) = \begin{bmatrix} 0 & \exp(\frac{i2\pi}{3}) \\ \exp(\frac{i4\pi}{3}) & 0 \end{bmatrix},$$

$$\Pi((2,1)) = \begin{bmatrix} 0 & \exp(\frac{i4\pi}{3}) \\ \exp(\frac{i2\pi}{3}) & 0 \end{bmatrix},$$

as expected. Apart from enumerating the irreps, we have also been able to come up with the fact that the only possible dimensions of irreps of G are 1, and q^t, and that there are exactly q^s irreps of dim-1, and $(p^r - 1)q^{(s-2t)}$ irreps of dim-q^t, where $1 \leq t \leq s$.

V. Future Work

We have so far been successful in classifying the group G as supersolvable and also in extracting its irreps. We wish to make use of the fact that any representation of a group G can be represented by direct decomposition of the irreps of G and be able to find the irreps of any subgroup, $H \leq G$. In other words, the irreps of G are guaranteed to describe the representation of every possible subgroup $H \leq G$, even uniquely, but only when decomposed in the right way. Having identified the irreps of H, we will use this fact to implement a quantum algorithm which we hope will efficiently solve the HSP in the case concerned.

VI. References

[1] P. W.Shor. Algorithms for quantum computation: discrete logarithms and factoring. 35^{th} *Annual Symposium on Fundamentals of Comp. Science*, pp 124-134, 1994

[2] D.Simon. On the power of quantum computation. 35^{th} *Annual Symposium on Fundamentals of Comp. Science*, pp 116-123, 1994

[3] A. Y. Kitaev. Quantum measurements and the Abelian stabilizer problem. *arXiv:quant-ph/9511026*, 1995.

[4] M. Ettinger and P. Høyer. On quantum algorithms for noncommutative hidden subgroups. *Adv, in Appl. Math.*, (25): pp 239-251, 2000.

[5] D. Bacon, A. M. Childs, and W. van Dam. From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semi-direct product groups. In 46^{th} *Annual Symposium on Fundamentals of Comp. Science*, pp 469-478, 2005.

[6] Y. Inui amd F. Le Gall. An efficient quantum algorithm for the hidden subgroup problem over a class of semi-direct product groups. *Quantum Information and Computation*, (5): pp 559-570, 2007.

[7] D. N. Gonçalves, R. Portugal, and C.M. M. Cosme. Solutions to the hidden subgroup problem on some metacyclic groups. 4^{th} *Workshop on Theory of Quantum Computation, Comm. and Cryptography.*, 2009

[8] C. Lomont. The Hidden Subgroup Problem - Review and Open Problems. *arXiv:quant-ph/0411037v1*, 2004.

[9] M. Hall Jr. The Theory of Groups. *American Mathematical Society, 2nd Edition*, 1999.

[10] J. P. Serre. Linear Representations of Finite Groups. *Springer*, 1977.

A Distance Based Algebra for Boolean Expression Queries over Large Networks

Arijit Khan
Computer Science, University of California, Santa Barbara
arijitkhan@cs.ucsb.edu

***Abstract*—A Boolean expression query consists of several negated and non-negated predicates connected by logical operators. Due to noise and the lack of fixed schema in information networks, it is often difficult to evaluate such queries. In this situation, the distance between two entities in the network provides a measure of how likely they are related to each other. Using this concept, we propose a novel distance-based algebra to define the top-k results for Boolean expression queries. We also provide a polynomial-time algorithm to efficiently answer Boolean-expression queries over large-scale networks. Empirical results show that our proposed technique can quickly find high-quality results in large networks.**

I. Introduction

In a wide array of disciplines, data can be modeled as an interconnected network of entities, and various attributes could be associated with both the entities and the relations among them. Examples of such systems include the World-Wide Web, social networks, biological networks, and online documents. Querying these information networks becomes important with a variety of applications including information extraction and pattern discovery.

In this paper, we consider Boolean expression queries over large networks. A Boolean expression query consists of several negated and non-negated predicates connected by logical operators. Hence, our objective is to identify the entities in the network that satisfy a given Boolean expression query. For example, let us consider the following query over *IMDB* movie network.

Example 1. *Find an actor who worked with both the directors "James Cameron" and "Michael Apted ", but did not work with the director "Steven Spielberg".*

The aforementioned query can be written as a Boolean expression query as follows. $Q_1 = P_1 \wedge P_2 \wedge \neg P_3$, where $P_1 \equiv$ actor worked with director "James Cameron", $P_2 \equiv$ actor worked with director "Michael Apted", $P_3 \equiv$ actor worked with director "Steven Spielberg". Observe that the query Q_1 consists of two non-negated and one negated predicates, and they are connected by conjunctive operators.

When the exact entity types, their names and relations are known in the dataset, such queries can be answered accurately, *e.g.,* using SQL. However, real-life information networks often lack standardized schemas. For example, in a movie network, there may be direct links between the actors and the director of a movie; or sometimes they can be indirectly connected via the corresponding movie. The exact entity names and the semantics of a link might also be unknown to the users. Therefore, it becomes difficult to answer such Boolean expression queries accurately over information networks.

If the schema and the semantics of the edge labels are not known, we argue that the distance between two entities in the network provides a measure of how likely they are related to each other [3]. For example, in our previous query, the required actor node should be close to both the director nodes, "J. Cameron" and "M. Apted", but far from "S. Spielberg". Using this concept, we propose a distance based **g**raph **algebra**, called gAlgebra, that outputs a ranked list of relevant nodes, for a given Boolean expression query.

We should also mention that the state-of-the-art ranked keyword search problem in graphs and XML documents [2] usually identifies the top-k nodes that are close to a given set of keywords. Therefore, the ranked keyword search is a specific case of gAlgebra, where several non-negated predicates are connected by only conjunctions.

II. Preliminaries

An information network can be modeled as a labeled, undirected graph $\mathcal{G} = (\mathbb{V}_\mathcal{G}, \mathbb{E}_\mathcal{G}, L_\mathcal{G})$ with node set $\mathbb{V}_\mathcal{G}$, edge set $\mathbb{E}_\mathcal{G}$, and label function $L_\mathcal{G}$, where (1) each node $u \in \mathbb{V}_\mathcal{G}$ represents an entity in the network, (2) each edge $e \in \mathbb{E}_\mathcal{G}$ denotes the relation between two entities, and (3) $L_\mathcal{G}$ is a function which assigns to each node $u \in \mathbb{V}_\mathcal{G}$ a set of labels $L_\mathcal{G}(u)$ from a finite alphabet. In practice, the node labels may represent the entity attributes, *e.g.,* name, type, etc. Next, we introduce fuzzy relation, which is the basic building block of gAlgebra.

Fuzzy Relation. For each node $u \in \mathbb{V}_\mathcal{G}$, we define a fuzzy relation $\mathbb{R}(u)$ as a set of tuples. Each tuple consists of a node $v \in \mathbb{V}_\mathcal{G} \setminus \{u\}$, and its membership function value $\mu(u, v)$.

$$\mu(u, v) = \alpha^{d(u,v)}; \quad \mathbb{R}(u) = \{\langle v, \mu(u, v)\rangle | v \in \mathbb{V}_\mathcal{G} \setminus \{u\}\}$$

Here, $d(u, v)$ is the distance (in hop counts) of v from u in the network, and α is a constant between 0 and 1. Therefore, the value of the membership function decreases exponentially with respect to the distance between two nodes. Observe that each predicate in a Boolean expression query may define multiple fuzzy relations, *e.g.,* the predicate, $P_1 \equiv$ who worked with director "James Cameron", defines

a set of fuzzy relations for each node u, where u's label, $L_{\mathcal{G}}(u)$ closely matches with "James Cameron (director)". Such a node u is referred to as a *defining node* for predicate P_1. Due to limitation of space, we shall assume that each query predicate has only one defining node.

The logical operators are defined below.

Negation. Let u be the defining node for predicate P_1. Then the corresponding fuzzy relation for the negated predicate $\neg P_1$ is denoted by $\mathbb{R}(\neg u)$, and is defined as follows.

$$\begin{aligned} \mu(\neg u, v) &= -\alpha^{d(u,v)}; \\ \mathbb{R}(\neg u) &= \{\langle v, \mu(\neg u, v)\rangle | v \in \mathbb{V}_{\mathcal{G}} \setminus \{u\}\} \end{aligned}$$

Conjunction. Let u and v be the defining nodes for predicates P_1 and P_2, respectively. The corresponding fuzzy relation for the conjunction, $P_1 \wedge P_2$ is denoted as $\mathbb{R}(u) \wedge \mathbb{R}(v)$.

$$\begin{aligned} \mu(u \wedge v, w) &= \mu(u, w) + \mu(v, w) \\ \mathbb{R}(u) \wedge \mathbb{R}(v) &= \{\langle w, \mu(u \wedge v, w)\rangle | w \in \mathbb{V}_{\mathcal{G}} \setminus \{u, v\}\} \end{aligned}$$

Disjunction. If u and v are the defining nodes for predicates P_1 and P_2, respectively, then the corresponding fuzzy relation for the disjunction, $P_1 \vee P_2$ is denoted as $\mathbb{R}(u) \vee \mathbb{R}(v)$.

$$\begin{aligned} \mu(u \vee v, w) &= \max\{\mu(u, w), \mu(v, w)\} \\ \mathbb{R}(u) \vee \mathbb{R}(v) &= \{\langle w, \mu(u \vee v, w)\rangle | w \in \mathbb{V}_{\mathcal{G}} \setminus \{u, v\}\} \end{aligned}$$

We are now ready to define our problem statement.

Problem Statement 1. *Given a network $\mathcal{G} = (\mathbb{V}_{\mathcal{G}}, \mathbb{E}_{\mathcal{G}}, L_{\mathcal{G}})$ and a Boolean expression query, find the top-k nodes with maximum membership function values.*

III. Algorithms for gAlgebra

We propose efficient algorithm to determine the top-k result nodes, for a given Boolean expression query.

Theorem 1. *Any Boolean expression query can be converted into an equivalent query that is in Disjunctive Normal Form (DNF),* i.e., *a disjunction of multiple clauses, where each clause is a conjunction of several predicates.*

Example 2. *Find the actor who worked with "Ben Kingsley", but not with "Marion Cotilland"; and also worked with either director "James Cameron", or "Christopher Nolan".*

Let us assume that nodes u, v, w and z represent "Ben Kingsley", "Marion Cotilland", "James Cameron" and "Christopher Nolan", respectively, in *IMDB*. Then the query can be expressed as $\mathbb{R}(u) \wedge \mathbb{R}(\neg v) \wedge (\mathbb{R}(w) \vee \mathbb{R}(z))$. The equivalent query in DNF will be $(\mathbb{R}(u) \wedge \mathbb{R}(\neg v) \wedge \mathbb{R}(w)) \vee (\mathbb{R}(u) \wedge \mathbb{R}(\neg v) \wedge \mathbb{R}(z))$. Given a Boolean expression query in DNF, we apply the following method based on the principle of Threshold Algorithm [1] that finds the top-k result nodes.

Disjunctive Threshold Algorithm. Assume that the set of clauses in the DNF query Q are given by $\mathbb{D} = \{D_1, D_2, \ldots, D_t\}$. For some clause $D_j \in \mathbb{D}$, let us denote by $\mathbb{V}_j$ the set of defining nodes for the non-negated predicates in D_j, and by $\neg\mathbb{V}_j$ the set of defining nodes for the negated predicates in D_j. We start performing breadth first search (BFS) in parallel from all $v \in \mathbb{V}_j$, $D_j \in \mathbb{D}$. For each node u encountered at the i-th step of the BFS, we calculate the aggregate value of the membership function of node u. Also, we compute $sum(i)$ as defined below. At some iteration i, if the top-k buffer is full, and $sum(i)$ is less than the smallest aggregate value of the membership function in the top-k buffer, we terminate the process, following the principle of Threshold Algorithm.

$$sum(i, j) = \alpha^i |\mathbb{V}_j|; \qquad sum(i) = \max_{D_j \in \mathbb{D}} sum(i, j)$$

IV. Experimental Results

We evaluated our techniques on the *YAGO* dataset ($1,768,773$ nodes; $2,199,056$ edges) from http://www.mpi-inf.mpg.de. The entity names were treated as node labels.

A. Efficiency

Figure 1 shows the high efficiency achieved by gAlgebra framework over large-scale networks. Each set of experiments was repeated for 100 Boolean expression queries. In Figure 1, we report the average running time. Specifically, our method finds the top-50 answers with 5 query predicates in less than 1 second.

B. Case Study

Q1. *Name the people who won both the Golden Globe and Grammy awards; but not the Academy award.*

The top-3 answers reported by our method is given in Figure 2. Observe that they are indeed all correct answers.

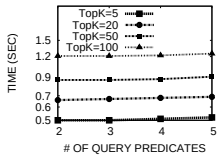


Figure 1: Efficiency

1	E . Murphy
2	E. Merman
3	R. Harris

Figure 2: Top-3 Results (**Q1**)

V. Conclusion

In this work, we introduced the novel problem of performing Boolean expression queries in information networks with unknown schema. In future, we may consider other relational algebra operations, *e.g.*, aggregation queries in network data.

References

[1] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[2] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[3] J. W. Kim and K. S. Candan. CP/CV: Concept Similarity Mining without Frequency Information from Domain Describing Taxonomies. In *CIKM*, 2006.

Identifying User Groups and Topic Evolution in Web Discussions

Theodore Georgiou
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA
teogeorgiou@cs.ucsb.edu

Manos Karvounis*
Yannis Ioannidis*
*Department of Informatics and Telecommunications
University of Athens
Athens, Greece
{manosk,yannis}@di.uoa.gr

***Abstract*—In this work we propose a new approach for identifying agreement and disagreement relationships between web forum participants and then, forming groups of agreeing discussants. Unlike previous similar efforts, our method is capable of extracting any number of groups and their pairwise relations. Moreover, to make the identified groups as coherent as possible, we propose a novel model for analyzing the evolution of topic, since, in many cases, agreement-disagreement relationships change as the discussion progresses, especially in large threads with abstract titles. Our approach is based exclusively on structural and temporal features and, thus, the textual content of posts is not required.**

I. INTRODUCTION

Web discussions constitute an integral part of the social web. The purpose of our work is to extract the rich network of agreement and disagreement relations formed during multiparty discussions on debate topics. To achieve this, we propose a method for identifying the pairwise relations between the discussants, and then create groups that share similar viewpoints. We firmly believe that the automated extraction of this network can greatly enhance our understanding of the dynamics of conversations in many ways:

- As a basis for deeper analysis of the extracted network (centrality, betweenness, bridge edges, etc.), to identify key characteristics of the discussion.
- As a pre-processing step for NLP techniques that mine the discussants' opinions. Clearly, it is much easier to extract these opinions if the agreement and disagreement relations are already known [3].
- As an overview mechanism, to present the discussant groups at a glance.

Although the core ideas of our method can be useful in many forms of multiparty debates, in this work we focus on discussions taking place in web forums. Web forum discussions unfold inside independent threads, characterized by their title. Like live discussions, a thread can easily deviate from its initial topic, and new relations between the discussants can be seamlessly created.

II. METHODOLOGY

Figure 1 shows an overview of the individual parts of our method, and how they are combined to produce the final results, i.e., groups of discussants and their relations.

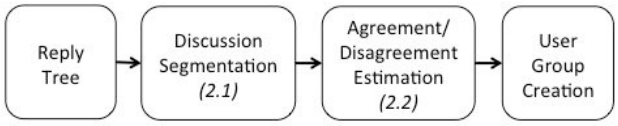


Figure 1: Flowchart of our method

The cornerstone of our methods is the extraction of the reply structure in the form of a tree (or forest, where applicable). The nodes are the individual posts inside the thread, and edges indicate that one post quotes and replies to the other. All the features we use are structural and temporal and are directly extracted from this tree.

To make the reply tree as complete as possible, we identify implicit reply connections using a trained SVM classifier. More specifically, we determine if an orphan post replies to the exact previous one. A post is considered orphan when it replies to other posts but doesn't explicitly quote them, a property that describes about 10% of the total number of a thread's posts. Our classifying technique is again structure-based, and has an accuracy of 75.4%.

Our next task is to partition the discussion into parts so that each contains one, specific topic. Clearly, new topics may mean new discussant relations, so we need to identify them and handle them separately. This is a novel addition since existing methods assume such relations to be static, a property rare in long discussions.

The last step is the group creation method, which consists of two tasks: (1) identification of discussants' pairwise relations and (2) clustering agreeing discussants into groups. These groups are created separately for each discussion sub-topic.

Our structure-only approach provides a number of advantages over NLP and hybrid approaches:

- It is independent from the language of the conversation.
- The text of the conversation need not be provided, thus solving privacy issues that frequently occur.

The main **contribution** of our work is the merging of topic evolution and agreement estimation into one concrete task aiming to extract the social network of online discussions. Moreover, we exploit the users' interaction patterns during a discussion to derive heuristics that only utilize structural features to achieve that.

A. Discussion Segmentation

Key observations for identifying when the discussion topic changes are the following:

- *Topics discussed in a threaded discussion do not cover more than one or two thematic areas [2]. On the other hand, sub-topics are numerous and usually clustered together [5] (parallel discussions are rare).*
- *Dense discussant activity can result in topic changes. The opposite is also applicable; when activity is sparse, the topic is more likely to remain the same [empirical observation].*
- *When the topic changes significantly, it is unlikely that discussants will reply to posts from the previous part of the discussion.*

To solve the discussion segmentation problem we need to identify areas in a thread where (a) local discussant activity (DA) is high and (b) local reply connections (RC) have short length. We measure the length between two connected posts as the number of other replies posted in between them (when viewed in temporal order).

We compute the derivatives of the two metrics DA and RC and search for points where a local DA maximum is *close* to a local RC minimum. In the example shown in Figure 2, that would be between posts 440 and 450. Since DA and RC are discrete variables the derivative is defined as:

$$f_x' = \frac{\Delta f}{\Delta x} = \frac{f_x - f_{x-1}}{x-(x-1)} = f_x - f_{x-1}$$

where x is the number of post.

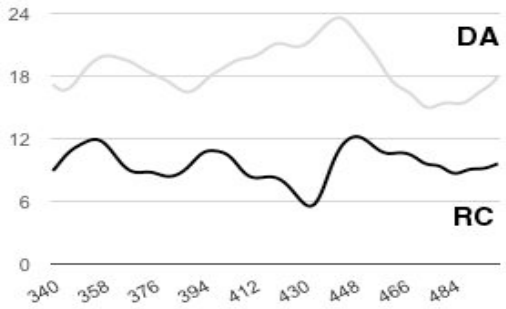


Figure 2: Normalized plots of the two functions in a real discussion

B. Agreement/Disagreement Discovery

For each pair of discussants, we create a feature vector and classify it as an agreement or disagreement using a manually trained classifier. The main observation supporting all the features we use is the following:

During a debate, the discussants devote most of their limited resources (time, effort) towards interacting with those that most threaten their arguments, i.e., those with whom they disagree.

Similar ideas have been used in previous works with significant success [4]. For each pair of users we define a 3-dimensional vector using the following features:

$$\text{target\_similarity}(u_i, u_j) = \text{cosine\_similarity}(\text{targets}(u_i), \text{targets}(u_j))$$

$$\text{reply\_focus}(u_i, u_j) = \max\left(\frac{replies(u_i,u_j)}{posts(u_i)}, \frac{replies(u_j,u_i)}{posts(u_j)}\right)$$

$$\text{reply\_coverage}(u_i, u_j) = \frac{\min\left(replies(u_i,u_j), replies(u_j,u_i)\right)}{\max\left(replies(u_i,u_j), replies(u_j,u_i)\right)}$$

where:

$$\text{targets}(u_i) = \left\langle \frac{replies(u_i,u_1)}{posts(u_i)}, \frac{replies(u_i,u_2)}{posts(u_i)}, \ldots, \frac{replies(u_i,u_{i-1})}{posts(u_i)}, 0, \frac{replies(u_i,u_{u+1})}{posts(u_i)}, \ldots, \frac{replies(u_i,u_n)}{posts(u_i)} \right\rangle$$

III. Evaluation

For a preliminary evaluation of our methods we utilized synthetic data that we created extending the models proposed by Kumar et al. [1] and Lin et al. [2] for simulating topic evolution, creating realistic reply tree structures and adding agreement/disagreement relations.

We were able to create many different discussion evolution structures and reply trees using a wide variety of value combinations on the models' parameters. Table 1 shows the precision/recall of the discussion segmentation method (evaluated separately) and precision of the final group creation method.

Method	Evaluation	
Discussion Segmentation	*Precision: **82%***	*Recall: **75%***
Group Creation	*Precision: **76%***	

The results seem very encouraging and on par with recent attempts on discussion segmentation and agreement-disagreement discovery.

IV. Future Directions

An essential task to complete the current work would be to create a large corpus of labeled real discussions to thoroughly evaluate and compare our methods with non-synthetic data. To the best of our knowledge such a dataset does not exist yet and would be an important contribution to the area of discussion mining.

References

[1] Ravi Kumar, Mohammad Mahdian and Mary McGlohon. 2010. Dynamics of conversations. In Proceedings of KDD '10.

[2] Lin, C., Yang, J., Cai, R., Wang, X., Wang, W. and Zhang L. 2009. Simultaneously Modeling Semantics and Structure of Threaded Discussions: A Sparse Coding Approach and Its Applications. In Proceedings of SIGIR '09.

[3] Chenhao Tan, Lillian Lee, Jie Tang, Long Jiang, Ming Zhou, and Ping Li. 2011. User-level sentiment analysis incorporating social networks. In Proceedings of KDD '11.

[4] Rakesh Agrawal, Sridhar Rajagopalan, Ramakrishnan Srikant, and Yirong Xu. 2003. Mining newsgroups using networks arising from social behavior. In Proceedings of WWW '03.

[5] Adams, P. H. and Martell, C. H. 2008. Topic Detection and Extraction in Chat. In Proceedings of ICSC '08.

Delay Tolerant Disaster Communication Using the One Laptop Per Child XO

Daniel Iland and Don Voita
University of California, Santa Barbara
Department of Computer Science
{iland,don}@cs.ucsb.edu

***Abstract*—In this paper, we describe the design, implementation, and testing of a peer-to-peer disaster messaging application for the One Laptop Per Child XO laptops. Our contributions are to implement a disaster communication network using an epidemic messaging scheme with cures on Ad-Hoc networks of OLPC XO laptops, extend the Ushahidi web application to accept messages originating from our OLPC peer-to-peer network, and allow the Ushahidi server to generate and distribute cures for each received message. We analyze and evaluate the network traffic, power consumption, and efficiency of this epidemic messaging scheme.**

***Keywords*-Delay Tolerant Networks; Disaster Networks; Epidemic Routing; OLPC; Ushahidi**

I. Introduction and Motivation

The importance of utilizing user generated reports in disaster response efforts has been recognized in recent years, as evidenced by the widespread adoption of web-based crisis mapping applications such as Ushahidi. Ushahidi is an open source application for receiving, curating, and displaying geographically tagged information. Helping individuals in the affected area to share information with each other and first responders can improve aid distribution, search and rescue efforts, damage assessment, regional security, and other humanitarian efforts.

II. Design and Implementation

A. Developing for OLPC

The OLPC project was designed with collaboration in mind. Software packages, called Activities, can be shared amongst users on an Ad-Hoc or Mesh network. An activity is a bundle of Python code and all associated libraries. Our activity is an OLPC activity that is shared between XO Laptops (XOs) using an Ad-Hoc network. When an XO joins a shared activity, it broadcasts messages and cures to other XOs participating in the activity. When a batch of messages and cures is received, they are serialized and stored on the devices flash memory. If an XO finds Internet connectivity, it uploads all uncured messages to the Ushahidi deployment, and downloads all known cures. Messages which have been 'cured' are no longer shared.

B. Implementation of Shared Messages

We use an epidemic messaging scheme for sharing of all messages and cures. Since we do not prioritize the transfer of cures over messages, our application uses passive cures.[1] While other delay-tolerant messaging schemes are more efficient, we feel in a disaster scenario it is important to maximize message delivery probability.[3]

To accomplish data synchronization, we utilized CausalDicts, shared data structures provided by the groupthink library. A CausalDict is a Python dict that is automatically shared between all XOs which have joined a shared activity. We use two CausalDict objects, one containing all messages and the other containing all cures. By using a hash derived from the message as the key, we ensure that all copies of a message are cured. Messages are tuples containing the title, content, category, location, time, and message hash.

OLPC designed the XO with an infrastructure free children sitting under the tree scenario as a primary use case. This sharing is primarily accomplished by two services, an interprocess communication system called D-Bus and a real-time communication framework called Telepathy. Telepathy allows applications running on different machines to communicate via an abstraction called tubes. Tubes are used to pass text and data, and can be implemented via a number of backend Connection Managers or protocols.

XOs use the Telepathy Salut protocol when connected to an Ad-Hoc network. On the network layer, Telepathy Salut uses multicast DNS (mDNS), and link local Extensible Messaging and Presence Protocol (XMPP) for device and service discovery. The XMPP protocol enables server-less messaging between clients via mDNS. Multi-user messaging, and therefore activity collaboration on the XO, is accomplished via an extension of XMPP called the Clique protocol.

C. Implementation of Ushahidi Plugin

When a device obtains internet access, it will upload each of its uncured messages to an Ushahidi server using Ushahidi's json API. The user-provided location string will be geolocated so the message can be placed on the map.

III. Testing and Characterization

A. Testing Approach

We tested how well our activity shared messages and cures, the network overhead involved in epidemic message passing between nodes, and power consumption of idle and active nodes. Our testbed consisted of 8 XO-1.5 laptops joined in an 802.11b/g Ad-Hoc network. We were able to directly monitor net battery discharge, by polling the value of an Accumulated Current Register (ACR) every 20 seconds. We used a laptop running Wireshark in Monitor Mode to capture the network traffic generated by each device.

We ran three tests, the first injecting 100 messages into the network, the second injecting 500 messages, and the third testing range and mobility indoors and outdoors to determine the maximum effective distance at which our activity shares data. In the third test, we used one mobile node to bridge two disconnected groups of nodes, one with Internet access. The mobile node received messages from group A, then was physically moved to be in range of group B. Messages from group A were delivered to group B, and uploaded to the Ushahidi server. Upon upload, the Ushahidi server generated and provided cures to the uploading node. These cures were passed back to our mobile node, which returned to the area of group A and delivered these cures.

B. Testing Results

Our experimental results show that our application works as expected, but is inefficient due to additional protocol overhead from Telepathy Salut. We also confirm the direct link between power consumption and wireless radio usage on XOs. Our tests show the wireless radio range of the OLPC is excellent (up to 350 meters) in areas with few obstructions, while still being acceptable (86-96 meters) in a campus setting with many obstructions and potential causes of multi-path fading (such as concrete buildings, trees, people, brick walls, and a building emergency power generator).

1) 100 and 500 Message Tests: Our experiments show messages account for the largest percentage of bytes, but less than 20% of overall packets. An unacceptably high percentage of both packet transmissions and throughput were used for Clique Overhead. Clique Overhead includes negotiation between clients to set up Telepathy tubes. These tubes allow for inter-machine communication. In our 100 message test, these overhead packets represent over 50% of the overall packets. In the 500 Message test, Clique Overhead is responsible for over 87% of the total packets, a striking increase. As observed in our 500 message test, the establishment and maintenance of inter-machine communication was extremely inefficient. In this test, of 788,884 total packets, 693,655 were Clique packets smaller than 120 bytes. Each packet contained 14 bytes or less of usable data, which represents at least 88.3% overhead per packet. We suspect this severe overhead is the main cause of only 5 of 8 nodes successfully receiving all messages.

2) Power Consumption: Over the course of both tests, we were able to directly monitor net battery discharge from each XOs battery. Figure 1 shows power consumption on the XO laptop named kangaroo. These results are representative of the other 7 XOs. As this graph shows, there is a strong correlation between the amount of data transferred and power consumption. Power consumption decreases as the number of bytes transmitted decreases. Therefore minimizing transmissions can play an important role in increasing network and device lifetime.[1] Our relatively inefficient epidemic messaging scheme likely uses more power than more conservative schemes, such as PRoPHET.[2]

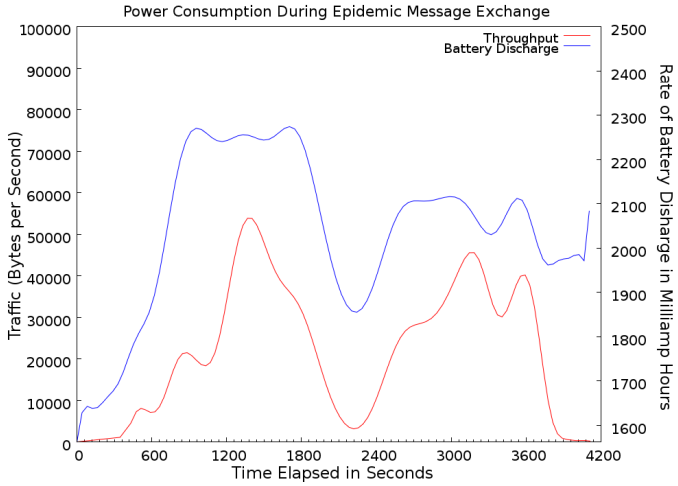


Fig. 1. 'kangaroo' receiving and forwarding 500 messages

[1]We observed relatively flat power consumption of these machines in an idle state (measured, but not shown).

IV. Future Work

Deploying on OLPC devices allows us to reach over 2.5 million students and teachers on 6 continents. As we proceed, other benefits include the open source nature of the project, and the broad spectrum of deployments, from dozens of XOs in rural villages in Australia and Africa to tens of thousands in Urban areas in Peru. Deploying a similar application to Android devices will allow us to reach hundreds of millions of users.

We have shown in our tests that power utilization is directly correlated with wireless radio utilization. One potential power-saving optimization would opportunistically increase the time between checks for an Internet connection, polling only when associated with an access point, when a neighbor informs an XO that it has recently had Internet access, or when the XO has uncured messages to send.

Switching from the OLPC's Telepathy based sharing system to a system utilizing UDP broadcasts will reduce overhead, enable interoperability with the Android-based system, and reduce unnecessary transmissions.

References

[1] K. A. Harras, K. C. Almeroth, and E. M. Belding-Royer. "Delay Tolerant Mobile Networks (DTMNs): Controlled Flooding in Sparse Mobile Networks." *In Proceedings of the IFIP-Networking*, Waterloo, Canada, May 2005.

[2] A. Lindgren, A. Doria, and O. Schel. "Probabilistic Routing in Intermittently Connected Networks." *SIGMOBILE Mobile Computing and Communications Review*, Volume 7, Issue 3, July 2003.

[3] L. Song and D. Kotz. "Evaluating Opportunistic Routing Protocols with Large Realistic Contact Traces." *In Proceedings of CHANTS*, Montreal, Canada, September 2007.

Mirror Mirror on the Ceiling: Flexible Wireless Links for Data Centers

Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li[§], Saipriya Kumar, Amin Vahdat[†], Ben Y. Zhao and Haitao Zheng
Department of Computer Science, U. C. Santa Barbara, USA
[§]Xi'an Jiaotong University, P. R. China [†]Google and U. C. San Diego, USA
{xiazhou,zengbin,yibo,saipriya,ravenben,htzheng}@cs.ucsb.edu, heatonlyb@gmail.com, vahdat@cs.ucsd.edu

I. Introduction

Modern distributed applications can run in data centers at massive scale. The bandwidth requirements of these applications can range from the relatively modest to substantial. Delivering such bandwidth comes at substantial cost for the switching infrastructure. As a result, recent efforts have investigated techniques to deploy more efficient data center network topologies.While these alternate topologies offer a range of benefits over the current state of the art, there are a number of inherent challenges with the deployment of *a*ny wired network technology. These include the cost, complexity, and inflexibility entailed by tens of thousands of fibers.

In this work, we focus on high-throughput, beamforming wireless links in the 60 GHz band. The unlicensed 60 GHz band provides multi-Gbps data rates and has small interference footprint. In particular, we identify two key limitations of existing 60 GHz proposals, and investigate the feasibility of 60 GHz 3D beamforming as a flexible wireless primitive in data centers. In 3D beamforming, a top-of-rack directional antenna forms a wireless link by reflecting a focused beam off the ceiling towards the receiver. This reduces its interference footprint, avoids blocking obstacles, and provides an indirect line-of-sight path for reliable communication. Such a system requires only beamforming radios readily available, and near perfect reflection can be provided by simple flat metal plates mounted on the ceiling of a data center.

While wired networks will likely remain the vehicle for the high-end distributed computing, we believe that efforts such as 3D beamforming can expand the applicability of wireless networking to a broader range of data center deployments.

II. 60 GHz Limitations

Existing designs adopt 60 GHz wireless technologies for several reasons. *First*, the 7GHz spectrum available in this band can deliver the multi-Gbps data rates required by data centers. *Second*, 60 GHz links operate at a high carrier frequency, which limits the interference they generate, and is highly beneficial to data centers with dense rack deployments. *Third,* 60 GHz links can use beamforming to enhance link rate and further suppress interference. Today, 60 GHz beamforming radios are readily available and affordable, either as directional (horn) antennas [4] or antenna arrays [3]. They use either mechanical or electronic mechanisms to achieve fine-grain directional control.

Link Blockage. Link blockage is a limiting factor for 60 GHz links. The 5mm wavelength of these links means that any object larger than 2.5 mm can effectively block signals or reflect them, producing multipath fading and degrading transmission rates. In today's data centers, this is problematic because racks are organized in a grid, and transceivers and antennas on one rack can easily block transmissions on another rack. This has led to current designs limiting themselves to connecting neighboring racks (see Figure 1(c)).

Radio Interference. Despite the use of beamforming to bound the transmission energy in a "narrow" direction, radio interference remains an issue for these systems. Radio design artifacts will still produce signal leaks outside of the intended direction. When placed in a dense rack formation, leakage produces harmful interference between nearby links and limits the density of concurrent links.

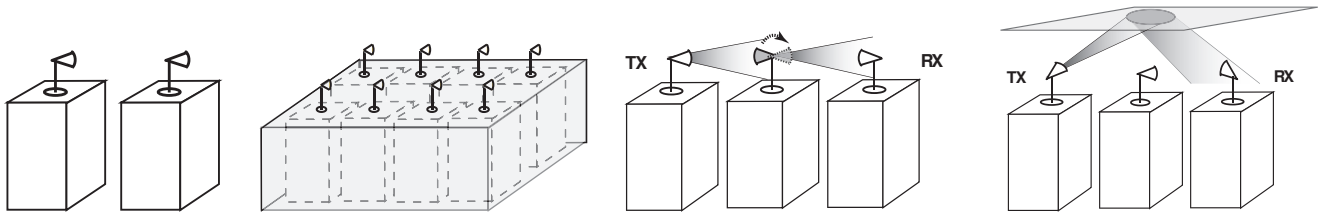
The spread of radio interference significantly limits the number of concurrent wireless links in a data center. One option is to separate the links in the frequency domain. But this reduces the per-link capacity, since the total available bandwidth is fixed across the frequency range. Alternatively, data center managers can increase the spacing between racks to reduce interference. But this leads to inefficient space and power usage, and weakens long-distance links.

III. Wireless 3D Beamforming

To address these limitations, we propose *3D beamforming*, a new beamforming approach that leverages ceiling reflections to connect racks wirelessly. An example is shown in Figure 1(d), where a transmitter bounces its signal off of the ceiling to the receiver. This creates an indirect line-of-sight path between the sender and receiver, bypassing obstacles and reducing interference footprint. To align its antenna for a transmission, the sender only needs to know the physical location of the receiver rack, and point to a position on the ceiling directly between the two racks.

3D beamforming requires three hardware components:

- *Beamforming Radios*: We reuse beamforming radios [3], [4] and adjust beam directions in both azimuth and elevation by placing the horn antennas on rotators. Existing rotators can achieve an accuracy of 0.006°-0.09°.
- *Ceiling Reflectors*: Reflectors act as *specular mirrors* to reflect signals. Our experiments confirm prior work show-



(a) Rack-based DC (b) Container-based DC (c) 2D Beamforming (d) 3D Beamforming

Fig. 1. Radio transceivers are placed on top of each rack (a) or container (b). Using 2D beamforming (c), transceivers need to forward traffic in multiple hops to non-neighboring racks. Using 3D beamforming (d), the ceiling reflects the signals from each sender to its receiver, avoiding multi-hop relays.

ing that flat metal plates offer perfect reflection without degrading energy or changing path loss characteristics.

- *Electromagnetic Absorbers*: We place electromagnetic absorbers near each antenna to prevent any local reflection and scattering. These absorbers require no maintenance.

A. Microbenchmark Results

Using detailed hardware experiments, we examine the key properties of 3D beamforming by comparing to 2D systems.

3D Beamforming Testbed. Our local testbed consists of two 60GHz beamforming radios from HXI [2], a 4ft×8ft metal reflector, and RF absorbers from ETS-Lindgren [1]. We test two types of reflectors: commercial-grade mirror-quality stainless steel plates and off-the-shelf cheap galvanized steel sheets from our local home improvement store. To assist with rapid experimentation, we mount the reflector vertically on a mobile platform that stands in parallel to a line connecting the center of the two radio transceivers. We vertically align platform using multiple hanging plumb-bobs. The corresponding ceiling height h is the perpendicular distance between the reflector and the line. To prevent reflected signals from producing more reflections at the receiver side, we place RF absorbers under the antenna. The absorber is a surface tiled with small pyramids 7.5cm thick. It does not block 3D transmit/reflection paths, but eliminates additional reflections. Finally, we manually calibrate the orientations of the horn antennas, using high precision laser pointers for guidance.

Property 1: Extended Link Connectivity. Our first experiment looks at link connectivity. Intuitively, using ceiling reflection, 3D beamforming will bypass obstacles in the horizontal plane, eliminating the antenna blockage problem of its 2D counterpart. More importantly, since ceiling reflectors should produce no loss, it should produce an indirect LOS path following the free-space propagation model.

Our measurement results match the model, confirming that both beamforming methods follow the free-space propagation model, and that the reflector introduces no energy loss. We also observe that both type of reflectors (commercial-grade and cheap steel plates) offer perfect reflection.

Property 2: Reduced Radio Interference. Our second experiment examines the interference footprint of both 2D and 3D beamforming. For both methods, we first set up a target transmission link X, then keep the transmitter intact and move the receiver around to measure link X's power emission map. We divide the measurement space into 0.3m×0.15m grids. In each grid, we rotate the receiver antenna to locate the direction with the maximum signal strength, subtract this strength by the receiver antenna gain, and use the result as the maximum interference that link X produces to this location.

We observe that for 2D beamforming, the directional wave still propagates freely in its beam direction, affecting other receivers along the path. The signal leakage also contributes to the level of interference. In contrast, 3D beamforming bounds the interference region to a much smaller area, and limits the impact of signal leakage. Please refer to [5] for more results.

Summary. Our testbed experiments verify that 3D beamforming largely addresses the main limitations of existing 60 GHz proposal. This means that we can connect most or all rack pairs using single hop links, thus maximizing bandwidth and eliminating forwarding delays. It also means a large number of links can be active in a small area without causing mutual interference and limiting performance.

B. Addressing Traffic Hotspots

We use network simulations to quantify 3D beamforming's ability to deliver additional bandwidth in data centers, and its advantages over its 2D counterpart. We use wireless links to cover traffic hotspots atop an existing wired network. Our findings are as following. *First*, in data centers with random traffic, 2D beamforming restricted to neighboring racks can only address a very limited ($\sim 3\%$) portion of traffic hotspots, compared to 100% for single hop 3D beamforming links. *Second*, for many scenarios with bursty traffic hotspots, using 3D beamforming links in conjunction with the existing wired network can generally reduce completion time by half or more. *Finally*, when sizable payloads are involved, *e.g.* 128MB, antenna rotation delays only contribute a small portion of the overall completion time, and much of that can be recovered using simple heuristics such as choosing radios that are closer to the desired transmission angle.

REFERENCES

[1] ETS-Lindgren. http://ets-lindgren.com/Absorbers.
[2] HXI Milimeter Wave Products. http://www.hxi.com/.
[3] SiBeam. http://sibeam.com/whitepapers/.
[4] HALPERIN, D., ET AL. Augmenting data center networks with multi-gigabit wireless links. In *Proc. of SIGCOMM* (2011).
[5] ZHOU, X., ET AL. Mirror mirror on the ceiling: Flexible wireless links for data centers. In *Proc. of SIGCOMM* (2012).

ImmuNet: Improved immunization of children through cellular network technology

Mariya Zheleva, Ceren Budak, Arghyadip Paul, Biyang Liu, Elizabeth M. Belding, and Amr El Abbadi
Department of Computer Science University of California, Santa Barbara
{mariya, cbudak, arghyadip, bliu, ebelding, amr}@cs.ucsb.edu

***Abstract*—Vaccination distribution and tracking, especially in remote areas in the developing world, is an important problem that can benefit from recent technological improvements in attaining more effective distribution mechanisms. In this paper we describe ImmuNet, a system that utilizes cellular network technology and allows rapid determination of immune status; reliable updates of vaccination records and quick targeted dissemination of vaccination availability in rural areas. In Summer 2012 our research team traveled for three weeks to the rural village of Macha, Zambia to deploy one of ImmuNet's modules and also to conduct interviews to gain understanding for immunization practices in remote rural areas.**

I. Problem motivation

While vaccination has lead to the successful eradication of a number of diseases in the developed world, developing countries still fall behind in implementing high-coverage immunization strategies. According to statistics from the World Health Organization, 1.5 million children died in 2010 from diseases preventable by vaccines. The highest percentage, 42%, of the cases were in the African region and 29% were in South East Asia. Most of the deaths were caused by Hib Meningitis, Measles, Pneumonia and Tetanus; all of these are diseases that can be prevented through vaccination. In many cases vaccines are available; however, vaccination schedules are difficult to enforce due to missing or incomplete information about vaccination availability and individuals who need immunization.

There is increased effort in incorporating Information and Communication Technology (ICT) to improve vaccination distribution to children and infants [1], [2]; however, such systems are often designed according to western models and operate under assumptions that do not hold in developing countries. One typical assumption is that personal immunization history can be kept easily by the health workers as they perform vaccinations. Our work with the immunization staff from the Mission Hospital at Macha, Zambia, unveils that immunization clinics are often understaffed and keeping personal records along with the day-to-day immunization routines is a very time consuming and challenging task.

Another common assumption is that digitalizing immunization records would help significantly improve the efficiency of storing and accessing patients data. Technology including such that facilitates digital data entry is just being introduced in the developing world. Thus, otherwise skillful health care staff often lacks basic technical skills such as typing. In fact, our work in Macha shows that at best people can hunt and peck and these are typically people involved with IT, not the health workers. As a result, digital data entry could turn into an additional hurdle that needs to be tackled by the already overworked health care staff. Furthermore, while adopting digital data entry, the immunization clinics still need to maintain their traditional practices of paper data entry, as this paper work is required on regular basis by the government officials. This results in doubling the effort put into keeping immunization history.

Challenges in distribution of vaccines are not only related to data entry and personal immunization history but also to outreach, especially to parents with children under the age of five, who do not attend school yet. Traditionally, information about vaccination schedules and availability is distributed through posters, word of mouth and in Macha, through the local community radio. Information is typically disseminated a few days or even weeks in advance, and as one of our interviewees from Macha shares, "It is very easy to forget which day exactly was vaccination day and miss an immunization". Thus, having a technology that is intuitive to people to help remind them for upcoming immunization events is of great importance.

To address these challenges and facilitate efficient distribution of vaccinations, we propose ImmuNet, a system that leverages cellular network technology and database tracking to keep individual immunization records. As most mandatory vaccinations occur between the age of 0 and 5 years, ImmuNet is designed for improved distribution of vaccinations to infants and children under the age of five. This focus, coupled with the specific vaccination practices in developing countries, poses unique challenges in the system design. To operate in rural areas with understaffed immunization personnel, the system enables collection of personal data in digital format that is submitted either by health workers or directly by the parents. By utilizing local cellular network infrastructure, the system provides prompt dissemination of information for vaccine availability and schedules in the form of text messages. ImmuNet can also record users' network association time and location and thus build patient interaction graphs to predict disease spread patterns and vaccination needs. Based on these interaction graphs, ImmuNet provides an alert system in the form of a heat-map that notifies health workers for high risk regions that need vaccinations.

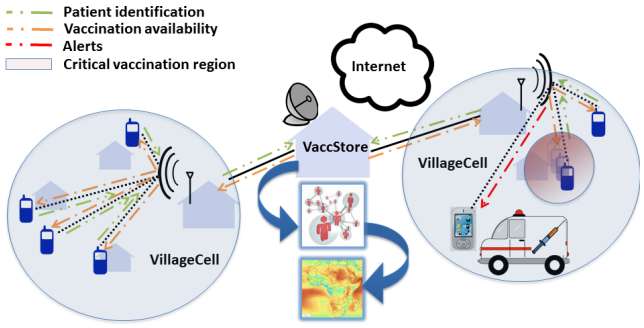


Fig. 1. ImmuNet Architecture: ImmuNet uses *VillageCell* as a cellular network platform and *VaccStore* as a database to collect patient identification, to distribute immunization information and to generate high-risk alerts.

II. System Overview

ImmuNet (Fig. 1) consists of two main components – a database called *VaccStore* and a local GSM network called *VillageCell*. While VaccStore is responsible for storing patient data, VillageCell is the network that provides connection with the patients. An engine, that runs on top of VaccStore is responsible for collecting patient immunization and mobility data from the network and sending alerts and reminders based on this data.

VaccStore. A key part of ImmuNet is the VaccStore database that stores personal biometric and identification data for each person, as available, and their immunization status. We implement VaccStore in MySQL, one of the most commonly used relational database management systems. A system that allows data to be submitted directly by the patients, inherently needs to support search on a noisy dataset. Such challenge cannot be handled with a traditional relational database solution. Therefore VaccStore extends this traditional paradigm to adapt to the requirements presented in the context of vaccination in developing regions. There are two main ways in which we extend the relational database paradigm: first we introduce a probabilistic data model to capture incomplete or uncertain data and, second, we incorporate basic diffusion models to capture the spread of diseases and influence and to create a *social* database.

VillageCell. We augment the original design of VillageCell, first proposed in [3], to a system that as of July 2012 covers an area of about 35 square kilometers in the village of Macha, Zambia. The system deployed in Macha, uses two RangeNetworks base stations that operate in the 1800 MHz band and each provides a coverage range of 5-7 kilometers. VillageCell uses open source and free software. Each base station runs an implementation of the GSM stack called OpenBTS. Calls are routed within or outside the local GSM network through Private Branch Exchange (PBX) software called FreeSwitch. To provide text messaging functionality in the system we use *smqueue*, which provides a store and forward SMS queue. This functionality is extremely important, as we expect that users will typically not be in range at all times, thus, their messages can be delivered in a delay tolerant fashion whenever they re-associate with VillageCell. The two VillageCell base stations deployed in Macha are connected over a wireless backhaul, which is already available through the village wireless infrastructure[1].

III. Interviews in Macha

During our visit in Macha in June/July 2012, we conducted interviews with 25 people from the area at the age of 17 to 48 years, to gain better understanding of cellphone usage in rural areas as well as people's perception of immunizations. Each interview was conducted in person between one interviewer and one interviewee. The interviewee's participation was voluntary and no material award was associated with the participation. To facilitate the interview process, we hired a woman from the community to introduce us to potential interviewees and help with translation. With interviewees' consent, 17 of the interviews were audio recorded.

The results from our interviews unveil that the benefit from immunizations is very well understood in the local community. All the people interviewed had been immunized when they were young and some of them perceive immunization as a "tradition" that needs to be followed, while others are keen to immunize their children, having experienced themselves the benefits of immunization in previous disease outbreaks, when they "did not suffer from the corresponding disease" because they were immunized. Only one of the participants who had a child under her care, did not do all recommended immunizations; all other participants had immunized all their children against all recommended diseases.

Cellphone usage too is widely adopted by people from the community as well as health workers. 24 of the 25 interviewees owned at least one cellphone and SIM card and all the interviewees were very accustomed to using basic functionality such as call and text messaging. 24 of the participants were excited about the idea for using their cellphone to receive SMS reminders for upcoming immunizations for their children.

IV. Conclusion

Incorporating technology for improved health care, especially in remote areas in the developing world, has had tremendous success over the last few years. ImmuNet continues in this direction by providing technology available at no additional end user cost for improved distribution of vaccines. Our work in the rural village of Macha, Zambia establishes distribution of vaccines as an important problem that can benefit from technological innovations. We are hopeful that by using a widespread technology such as cellphones, our system has great potential to improve the distribution of vaccines in the developing world.

References

[1] "http://www.childcount.org/."
[2] J. G. Jørn Klungsøyr, "Using mobile phones to track immunizations," *Wireless Health Organization*, 2010.
[3] A. Anand, V. Pejovic, E. M. Belding, and D. L. Johnson, "VillageCell: Cost effective cellular connectivity in rural areas," ser. ICTD, Atlanta, Georgia, March, 2012.

[1] http://www.machaworks.org/en/linknet.html

Breaking the Loop: Leveraging Botnet Feedback for Spam Mitigation

Gianluca Stringhini, Manuel Egele, Christopher Kruegel, and Giovanni Vigna
Computer Security Lab, University of California, Santa Barbara
{gianluca, maeg, chris, vigna}@cs.ucsb.edu

***Abstract*—In this paper, we propose a novel technique to fight spam. This technique leverages the observation that existing spamming botnets leverage the feedback provided by mail servers to tweak their operations, and send spam more effectively. We show that, by sending wrong feedback to those clients that are detected as spammers, we are able to impact the efficiency of future spam campaigns sent by that same botnet.**

I. Introduction

Email spam, or *unsolicited bulk email*, is one of the major open security problems of the Internet. Accounting for more than 77% of the overall world-wide email traffic [2], spam is annoying for users who receive emails they did not request, and it is damaging for users who fall for scams and other attacks. Also, spam wastes resources on SMTP servers, which have to process a significant amount of unwanted emails [6]. A lucrative business has emerged around email spam, and recent studies estimate that large affiliate campaigns generate between $400K and $1M revenue per month [1].

Nowadays, more than 85% of worldwide spam is carried out by botnets. Spamming botnets typically use *template-based spamming* to send out emails [3]–[5]. With this technique, the botnet Command and Control infrastructure tells the bots what kind of emails to send out, and the bots relay back information about the delivery as they received it from the SMTP server. This server feedback is an important piece of information to the botmaster, since it enables him to monitor if his botnet is working correctly.

Of course, a legitimate sender is also interested in information about the delivery process. However, she is interested in different information compared to the botmaster. In particular, a legitimate user wants to know whether the delivery of her emails failed (e.g., due to a typo in the email address). In such a case, the user wants to correct the mistake and send the message again. In contrast, a spammer usually sends emails in batches, and typically does not care about sending an email again in case of failure.

Nonetheless, there are three main pieces of information related to server feedback that a rational spammer is interested in:

1) Whether the delivery failed because the IP address of the bot is blacklisted.
2) Whether the delivery failed because of specific policies in place at the receiving end (e.g., greylisting).
3) Whether the delivery failed because the recipient address does not exist.

In all three cases, the spammer can leverage the information obtained from the mail server to make his operation more effective and profitable. In the case of a blacklisted bot, he can stop sending spam using that IP address, and wait for it to be whitelisted again after several hours or days. Empirical evidence suggests that spammers already collect this information and act accordingly [5]. If the recipient server replied with an SMTP non-critical error (i.e., the ones used in greylisting), the spammer can send the email again after some minutes to comply with the recipient's policy.

The third case, in which the recipient address does not exist, is the most interesting, because it implies that the spammer can *permanently* remove that email address from his email lists, and avoid using it during subsequent campaigns. Recent research suggests that bot feedback is an important part of a spamming botnet operation. For example, Stone-Gross et al. [5] showed that about 35% of the email addresses used by the *Cutwail* botnet were in fact non-existent. By leveraging the server feedback received by the bots, a rational botmaster can get rid of those non-existing addresses, and optimize his spamming performance significantly.

II. Providing False Responses to Spam Emails

Based on these insights, we want to study how we can manipulate the SMTP delivery process of bots to influence their sending behavior. We want to investigate what would happen if mail servers started giving erroneous feedback to bots. In particular, we are interested in the third case explained in Section I, since influencing the first two pieces of information has only a limited, short-term impact on a spammer. However, if we provide false information about the status of a recipient's address, this leads to a double bind for the spammer: on the one hand, if a spammer considers server feedback, he will remove a valid recipient address from his email list. Effectively, this leads to a reduced number of spam emails received at this particular address. On the other hand, if the spammer does not consider server feedback, this reduces the effectiveness of his spam campaigns since emails are sent to non-existent addresses. In the long run, this will significantly degrade the freshness of his email lists and reduce the number of successfully sent emails. In the following, we discuss how we can take advantage of this situation.

As a first step, we need to identify that a given IP address is in fact a spambot. To this end, a mail server can either use traditional, IP-based blacklists or use alternative spam detection techniques. Once we have identified a bot, a mail server

can (instead of closing the connection) start sending erroneous feedback to the bot, which will relay this information to the Command and Control infrastructure. Specifically, the mail server could, for example, report that the recipient of that email does not exist. By doing this, the email server would lead the botmaster to the lose-lose situation discussed before. For a rational botmaster, we expect that this technique would reduce the amount of spam the email address receives.

III. Evaluation

To investigate the effects of wrong server feedback to bots, we set up the following experiment. We ran 32 malware samples from four large spamming botnet families (*Cutwail*, *Lethic*, *Grum*, and *Bagle*). We picked these families because they were responsible for the vast majority of the worldwide spam at the time of our experiment. We ran the samples in a controlled environment, and redirected all of their SMTP activity to an email server under our control. We configured this server to report that *any* recipient of the emails the bots were sending to was non-existent. Furthermore, we also used firewall rules to limited the outgoing network traffic and contain unforeseen side-effects of executing the bots in our environment. To the best of our knowledge, no malicious traffic left our experimental network.

To assess whether the different botnets stopped sending emails to those addresses, we leveraged a *spamtrap* under our control. A spamtrap is a set of email addresses that do not belong to real users, and, therefore, collect only spam mails. To evaluate our approach, we leverage the following idea: if an email address is successfully removed from an email list used by a spam campaign, we will not observe the same campaign targeting that address again. We define as a spam campaign the set of emails that share the same URL templates in their links, similar to the work of Xie et al. [7]. While there are more advanced methods to detect spam campaigns [4], the chosen approach leads to sufficiently good results for our purposes.

We ran our experiment for 73 days, from June 18 to August 30, 2011. During this period, our mail server replied with false server feedback for 3,632 destination email addresses covered by our spamtrap, which were targeted by 29 distinct spam campaigns. We call the set of campaigns C_f and the set of email addresses S_f. Of these, five campaigns never targeted the addresses for which we gave erroneous feedback again. To estimate the probability P_c that the spammer running campaign c in C_f actually removed the addresses from his list, and that our observation is not random, we use the following formula:

$$P_c = 1 - (1 - \frac{n}{t_f - t_b})^{t_e - t_f},$$

where n is the total number of emails received by S_f for c, t_f is the time at which we first gave a negative feedback for an email address targeted by c, t_b is the first email for c which we ever observed targeting our spam trap, and t_e is the last email we observed for c. This formula calculates the probability that, given a certain number n of emails observed for a certain campaign c, no email was sent to the email addresses in S_f *after* we sent a poisoned feedback for them. We calculate P_c for the five campaigns mentioned above. For three of them, the confidence was above 0.99. For the remaining two, we did not observe enough emails in our spamtrap to be able to make a final estimate.

To assess the impact we would have had when sending erroneous feedback for all the addresses in the spamtrap, we look at how many emails the whole spamtrap received from the campaigns in C_f. In total, 2,864,474 emails belonged to campaigns in C_f. Of these, 550,776 belonged to the three campaigns for which we are confident that our technique works and reduced the amount of spam emails received at these addresses. Surprisingly, this accounts for 19% of the total number of emails received, indicating that this approach could have impact in practice.

IV. Conclusions

We presented a novel approach to mitigating spam, based on manipulating the server feedback. We acknowledge that these results are preliminary and provide only a first insight into the large-scale application of server feedback poisoning. Nevertheless, we are confident that this approach is reasonable since it leads to a lose-lose situation for the botmaster. We argue that the uncertainty about server feedback introduced by our method is beneficial since it reduces the amount of information a spammer can obtain when sending spam.

We realize that cybercriminals might adapt, and try to detect poisoned feedback, for example by discarding any response coming from a server that always reports that email addresses do not exist. Having cybercriminals adapt is a common problem in the arms race between researchers and miscreants. However, discarding any feedback would still affect botmasters in a negative way, making our approach still effective.

References

[1] C. Kanich, N. Weaver, D. McCoy, T. Halvorson, C. Kreibich, K. Levchenko, V. Paxson, G. Voelker, and S. Savage. Show Me the Money: Characterizing Spam-advertised Revenue. *USENIX Security Symposium*, 2011.

[2] Kaspersky Lab. Spam Report: April 2012. https://www.securelist.com/en/analysis/204792230/Spam_Report_April_2012, 2012.

[3] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[4] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. botnet Judo: Fighting Spam with Itself. In *Symposium on Network and Distributed System Security (NDSS)*, 2010.

[5] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna. The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[6] B. Taylor. Sender reputation in a large webmail service. In *Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2006.

[7] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. *SIGCOMM Comput. Commun. Rev.*, 38, August 2008.

Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner

Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna
University of California, Santa Barbara
{adoupe, cavedon, chris, vigna}@cs.ucsb.edu

***Abstract*—Black-box web vulnerability scanners are a popular choice for finding security vulnerabilities in web applications. Unfortunately, black-box tools suffer from a number of limitations, particularly when interacting with complex applications that have multiple actions that can change the application's state.**

We propose a novel way of inferring the web application's internal state machine *from the outside*—that is, by navigating through the web application, observing differences in output, and producing a model representing the web application's state.

We utilize the inferred state machine to drive a black-box web application vulnerability scanner.

I. Introduction

Web applications are the most popular way of delivering services via the Internet. The complexity of modern web applications, along with the many different technologies used in various abstraction layers, are the root cause of vulnerabilities in web applications.

One method of vulnerability discovery is called *black-box* testing, as the application is seen as a sealed machine with unobservable internals. Black-box approaches are able to perform large-scale analysis across a wide range of applications.

Classical black-box web vulnerability scanners crawl a web application to enumerate all reachable pages and then fuzz the input data to trigger vulnerabilities. However, this approach ignores a key aspect of modern web applications: Any request can change the state of the web application.

In the most general case, the state of the web application is any data (database, filesystem, time) that the web application uses to determine its output.

Because a black-box web vulnerability scanner will never detect a vulnerability on a page that it does not see, scanners that ignore a web application's state will only explore and test a fraction of the web application.

In this paper, we propose to improve the effectiveness of black-box web vulnerability scanners by increasing their capability to understand the web application's internal state.

II. Motivation

Crawling modern web applications means dealing with the web application's changing state. Previous work in detecting workflow violations [1]–[4] focused on navigation, where a malicious user can access a page that is intended only for administrators. This unauthorized access is a violation of the developer's intended work-flow of the application.

We wish to distinguish a navigation-based view of the web application, which is simply derived from crawling the web application, from the web application's internal state machine. To illustrate this important difference, we will use a small example.

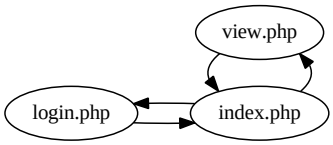


Fig. 1. Navigation graph of a simple web application.

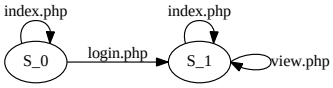


Fig. 2. State machine of a simple web application.

Consider a simple web application that has only three pages: `index.php`, `login.php`, and `view.php`. The `view.php` page is only accessible after the `login.php` page is accessed. There is no logout functionality. A client accessing this web application might make a series of requests: ⟨`index.php`, `login.php`, `index.php`, `view.php`, `index.php`, `view.php`⟩

Analyzing this series of requests from a navigation perspective creates a navigation graph, shown in Figure 1. This graph shows which page is accessible from every other page, based on the navigation trace. However, the navigation graph does not represent the information that `view.php` is only accessible after accessing `login.php`, or that `index.php` has changed after requesting `login.php` (it includes the link to `view.php`).

We are interested in how the requests we make influence the web application's internal state machine. The simple web application described previously has the internal state machine shown in Figure 2. The web application starts with the internal state `S_0`. Arrows from a state show how a request affects the web application's internal state machine. In this example, in the initial state, `index.php` does not change the state of the application, however, `login.php` causes the state to transition from `S_0` to `S_1`. In the new state `S_1`, both `index.php` and `view.php` do not change the state of the web application.

Now the question becomes: "How does knowledge of the web application's state machine (or lack thereof) affect a black-box web vulnerability scanner?" The scanner's goal is to find vulnerabilities in the application, and to do so it must fuzz

as many execution paths of the server-side code as possible. Consider the simple application described in Figure 2. In order to fuzz as many code paths as possible, a black-box web vulnerability scanner must fuzz the `index.php` page in both states `S_0` and `S_1`, since the code execution of `index.php` can follow different code paths depending on the current state (more precisely, in state `S_1`, `index.php` includes a link to `view.php`, which is not present in `S_0`).

III. Inferring the State Machine

Inferring a web application's state machine requires the ability to detect when the state of the web application has changed.

The key insight of our state-change algorithm is the following: We detect that the state of the web application has changed when we make an identical request and get a different response. This is the only externally visible effect of a state-change: Providing the same input causes a different output.

Using this insight, our state-change detection algorithm works as follows: (1) Crawl the web application sequentially, making requests based on a link in the previous response. (2) Assume that the state stays the same, because there is no evidence to the contrary. (3) If we make a request identical to a previous request and get a different response, then we assume that some request since the last identical request changed the state of the web application.

The state-change detection algorithm allows us to infer when the web application's state has changed, yet four other techniques are necessary to infer a state machine.

Clustering similar pages. We want to group together pages that are similar to detect when a response has changed.

Before we can cluster pages, we model them using the links present on the page. The intuition here is that the links describe *how* the user can interact with the web application. Therefore, changes to what a user can do (new or missing links) indicate when the state of the web application has changed.

Determining the state-changing request. The state-change detection algorithm only says that the state has changed, however we need to determine *which* request actually changed the state. When we detect a state change, we have a temporal list of requests with identical requests at the start and end. One of the requests in this list changed the state. We use a heuristic to determine which request changed the state.

Collapsing similar states. The state-change detection algorithm detects only when the state has changed, however, we need to understand if we returned to a previous state. This is necessary because if we detect a state change, we want to know if this is a state we have previously seen or a brand new state. We reduce this problem to a graph coloring problem, where the nodes are the states and an edge between two nodes means that the states cannot be the same. We add edges to this graph by using the requests and responses, along with rules to determine when two states cannot be the same. After the graph is colored, states that are the same color are collapsed into the same state.

Navigating. We have two strategies for crawling the web application.

First, we always try to pick a link in the last response. The rational behind choosing a link in the last response is that we emulate a user browsing the web application. In this way, we are able to handle multi-step processes, such as previewing a comment before it is committed.

Second, for each state, we make requests that are the least likely to change the state of the web application. The intuition here is that we want to first see as much of a state as possible, without accidentally changing the state, in case the state change is permanent.

IV. State-Aware Fuzzing

After we crawl the web application, our system has inferred, as much as possible, the web application's state machine. We use the state machine information, along with the list of request–responses made by the crawler, to drive a state-aware fuzzing of the web application.

To fuzz the application in a state-aware manner, we need the ability to reset the web application to the initial state (the state when we started crawling). We do not use this ability when crawling, only when fuzzing.

Our state-aware fuzzing starts by resetting the web application to the initial state. Then we go through the requests that the crawler made, starting with the initial request. If the request does not change the state, then we fuzz the request as a typical black-box scanner. However, if the request is state-changing, we follow a simple algorithm: We make the request, and if the state has changed, traverse the inferred state machine to find a series of requests to transition the web application to the previous state. If this does not exist, or does not work, then we reset the web application to the initial state, and make all the previous requests that the crawler made. This ensures that the web application is in the proper state before continuing to fuzz.

V. Conclusion

We have described a novel approach to inferring, as much as possible, a web application's internal state machine. Using this approach, our crawler is able to crawl—and thus fuzz—more of the web application than a classical state-agnostic crawler. We believe our approach to detecting state change by differences in output for an identical response is valid and should be adopted by all black-box tools that wish to understand a web application's internal state machine.

References

[1] D. Balzarotti, M. Cova, V. Felmetsger, and G. Vigna, "Multi-module Vulnerability Analysis of Web-based Applications," in *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, 2007, pp. 25–35.

[2] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications," in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2007)*, 2007, pp. 63–86.

[3] X. Li and Y. Xue, "BLOCK: A Black-box Approach for Detection of State Violation Attacks Towards Web Applications," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2011)*, Orlando, FL, December 2011.

[4] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, "Toward Automated Detection of Logic Vulnerabilities in Web Applications," in *Proceedings of the USENIX Security Symposium*, Washington, DC, August 2010.

QuadMat: An Efficient and Extensible Sparse Matrix Structure

Adam Lugowski
Department of Computer Science
University of California, Santa Barbara
alugowski@cs.ucsb.edu

John R. Gilbert
Department of Computer Science
University of California, Santa Barbara
gilbert@cs.ucsb.edu

***Abstract*—We propose a quaternary tree-based data structure for sparse matrices, QuadMat. This structure allows more efficient memory hierarchy usage than existing sparse matrix structures.**

I. Introduction

Linear algebraic primitives are the basis for a large number of High-Performance Computing tasks. They constitute the ground work of many simulation packages, enable efficient data analysis, and even form the basis of graph algorithms. The Knowledge Discovery Toolbox (KDT) [3] is a high-performance graph analysis framework based on linear algebraic primitives on sparse structures.

The benefits of linear algebraic primitives over traditional graph algorithm frameworks boil down to superior memory utilization and superior parallelism. Traditional algorithms are data-driven with fine, irregular and unstructured memory accesses. This means poor locality of reference, unpredictable communication in a distributed setting, and code dominated by latency. Linear algebraic primitives, on the other hand, have fixed communication patterns with operations on matrix blocks which exploit the memory hierarchy and are dominated by available bandwidth.

KDT exploits these benefits to build a graph analysis framework that can scale from laptops to supercomputers running on thousands of cores. KDT's graph algorithms are built on extensible sparse matrix and vector structures using algorithms such as matrix-matrix and matrix-vector multiplication, element-wise operations, and reductions.

This paper focuses on the sparse matrix structure itself. We identify weaknesses of existing sparse matrix approaches and propose a new quaternary-tree based structure called QuadMat which addresses those weaknesses.

II. Current Approaches and Related Work

At its heart, a sparse matrix A is a set of tuples $(i, j, A_{i,j})$. An unordered set of tuples makes for inefficient algorithms, so there are many schemes to compress and rearrange these tuples to improve space complexity, time complexity, and locality of reference. The most commonly used schemes are the Compressed Sparse Columns (CSC) and its transpose, Compressed Sparse Rows (CSR).

CSC groups all tuples in the same column together, then sorts them by row. There is a single column index for every column, shared by all elements in that column. This approach makes it trivial to scan down columns of a matrix.

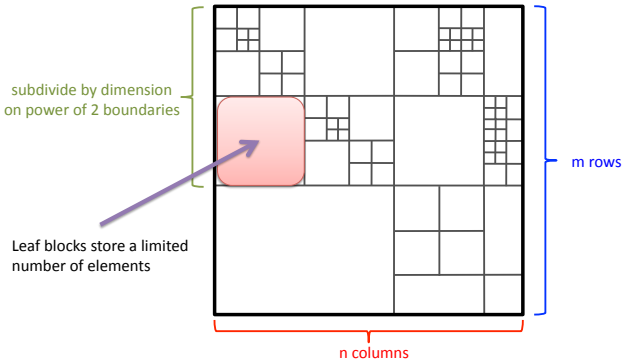


Fig. 1: A visual representation of a QuadMat quaternary tree decomposition of a matrix. Each intermediate block has 4 sub-blocks, and each leaf blocks stores a limited number of matrix elements. Splits occur on powers of 2 rows/columns.

KDT's backend, the Combinatorial BLAS [1], uses a Doubly-Compressed Sparse Column (DCSC) sparse matrix layout. DCSC further compresses CSC's column index array, which is advantageous in cases where there are fewer elements than columns. This frequently happens when a matrix is decomposed along its dimensions.

Blocking can make subdivision easier, both to exploit the memory hierarchy and to ease parallelization. Compressed Sparse Blocks [2] is a dense array of sparse blocks, where each block's elements are sorted in Morton Z-Order [5] for both cache-friendliness and to allow further splitting. CSB can perform sparse matrix-vector ($A \times V$) and sparse matrix transpose-vector ($A^T \times V$) multiplication in equal time.

Finally [4] shows that even a basic quad-tree matrix implementation can speed up sparse matrix-matrix multiplication.

A. Motivation

The sparse matrix structures in use today suffer from several disadvantages:

1) Reads are efficient in one direction ("down the columns"), but very inefficient in the other ("across the rows").

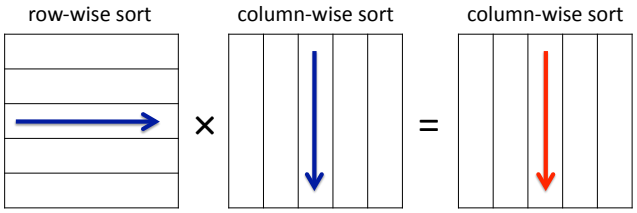


Fig. 2: QuadMat's changeable sort direction allows efficient reads of both operands and a convenient construction of the result.

2) Changes to the matrix structure require making a new copy of the entire matrix.
3) CSC mandates a form of data compression which is essentially run-length encoding. Many more sophisticated compression schemes are now available.

Read direction is important for operations like reduction and matrix-matrix or matrix-vector multiplication. The difficulty of changing the structure of the matrix means that the structure of the result must be known before it can be filled. These two constraints result in matrix-matrix multiplication codes that are run twice (once to find the structure of the answer, again to fill it in), or use an auxiliary structure which introduces additional overhead. The final point is important because linear algebraic primitives are memory-bandwidth bound and better compression can result in better use of available bandwidth.

III. QuadMat: Blocked Triples Storage

Our proposed scheme is a matrix that is recursively subdivided into quadrants, by dimension, in a quad-tree fashion as illustrated by Figure 1. Each leaf node has a minimal and maximal size. The leaf would present a tuple interface, but its internal storage can be nearly anything. Each leaf's sort direction (along rows, columns, or arbitrary) can be changed at any time to match the read direction needed. Nodes (internal or leaf) can be swapped out without affecting the rest of the structure.

This proposed scheme solves the problems stated above in the following ways.

A runtime-changeable sort direction allows efficient reads in any direction, as illustrated by Figure 2. Both operands are read in a streaming fashion, and the output is written in a streaming fashion. The sort is local to a leaf block, so it takes $O(k\ log\ k)$ time for a block with k elements. This means the total structure can be sorted in $O(n\ log\ k)$ time, not $O(n\ log\ n)$. Finally, the sort is done lazily and is likely to be amortized. Note that the output has the same sort direction as the right operand, meaning that repetitive operations (such as iterative solvers) will not require a resort in successive iterations.

Elements can be added to the matrix at runtime by swapping (small) individual leaf blocks. This does not require a complete structural rebuild.

A leaf node can store its data in any way it chooses. A basic $(i, j, A_{i,j})$ triples store can be very practical due to the nature of the quad-tree: each level in the tree implies a bit of both indices. In other words, depending sparsity, leaves may store only 16-bit offsets instead of full 32- or 64-bit indices. Additionally a fully dense node may store elements in a 2D array with a bitmask specifying empty elements. One can also envision CSC/CSR leaves, leaves that use ZIP or other types of modern compression, and even generator leaves that store nothing but a generation algorithm (useful for regular structures like an identity or grid matrix). The beauty of this approach is that multiple leaf types can be mixed together in the same structure, and even changed at runtime (so a sparse matrix can become dense, for example).

Block heterogeneity depends on a common interface. Leaf blocks will need to provide the following operations:

- Iterator(Sort) - provide an iterator that will efficiently traverse the block in the given direction (rows or columns, may require a resort of the data)
- Transpose - for a tuple store a trivial constant-time operation that means simply swapping the row and column array pointers
- Subdivide - present the leaf node as an internal node backed by the leaf node's data. Useful to virtually subdivide a large but sparse node to interface with small but dense nodes.

IV. Implementation

We are in the early stages of an implementation of this scheme. The project will initially target shared-memory systems with Intel Threading Building Blocks for parallelism. QuadMat is intended to become a shared-memory-only backend for KDT. It will also provide threading support to the Combinatorial BLAS to move it from a pure MPI system to a hybrid one to take better advantage of modern clusters.

References

[1] A. Buluç and J.R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *The International Journal of High Performance Computing Applications*, 25(4):496–509, 2011.

[2] Aydın Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-First ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, Calgary, Canada, August 2009.

[3] Adam Lugowski, David Alber, Aydın Buluç, John R. Gilbert, Steve Reinhardt, Yun Teng, and Andrew Waranis. A flexible open-source toolbox for scalable complex graph analysis. In *Proceedings of the Twelfth SIAM International Conference on Data Mining (SDM12)*, pages 930–941, April 2012.

[4] Ivan Simecek. Sparse matrix computations using the quadtree storage format. In *Proceedings of the 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '09, pages 168–173, Washington, DC, USA, 2009. IEEE Computer Society.

[5] David Wise. Ahnentafel indexing into morton-ordered arrays, or matrix locality for free. In Arndt Bode, Thomas Ludwig, Wolfgang Karl, and Roland Wismller, editors, *Euro-Par 2000 Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 774–783. Springer Berlin / Heidelberg, 2000.

Revolver: Detecting Evasion Attacks in Malicious JavaScript Code

Alexandros Kapravelos*, Yan Shoshitaisvili*, Marco Cova†, Christopher Kruegel* and Giovanni Vigna*
* UC Santa Barbara
{*kapravel,yans,chris,vigna*}*@cs.ucsb.edu*
† University of Birmingham
m.cova@cs.bham.ac.uk

***Abstract*—In recent years, attacks targeting web clients, in particular, web browsers and their plugins, have become a prevalent threat. Attackers deploy web pages that contain exploit code, typically written in HTML and JavaScript, and use them to compromise unsuspecting victims.**

A number of approaches have been proposed to detect malicious web pages and protect web users. However, these techniques are not perfect and attackers have found ways to circumvent them. In particular, attackers routinely tweak their exploit code to create new variants that evade commonly-used defensive tools.

In this paper, we present *Revolver*, a novel approach to identifying evasion attempts in malicious JavaScript code. *Revolver* leverages the observation that two scripts that are similar should be classified in the same way by malware detectors (either both are malicious or benign); differences in the classification indicate that one script may contain code designed to evade the detector tool. Thus, *Revolver* uses efficient techniques to identify similarities between large numbers of JavaScript programs (despite their use of obfuscation techniques, such as packing, polymorphism, and dynamic code generation), and to automatically interpret their differences.

I. Introduction

In the last several years, we have seen web-based malware—malware distributed over the web, exploiting vulnerabilities in web browser and their plugins—become a prevalent threat. In a recent report, Microsoft indicated that it detected over 25 million web-based exploits in the first quarter of 2011 alone. In particular, drive-by-download attacks are the method of choice for attackers to compromise and take control of victim machines. At the core of these attacks are pieces of malicious HTML and JavaScript code that redirect unsuspecting visitors to rogue web sites and launch browser exploits.

A number of techniques have been recently proposed to detect the code used in drive-by-download attacks. A common approach is the use of honeyclients (specially instrumented browsers) that visit a suspect page and extract a number of features that can help in determining if this page is benign or malicious. Such features can be based on static characteristics of the examined code [1], on specifics of its dynamic behavior [2], or on a combination of static and dynamic features.

Unfortunately, honeyclients are not perfect and attackers have found ways to evade them [3]. For example, malicious web pages may be designed to launch an exploit only after they have verified that the current visitor is a regular user, rather than an automated detection tool. A web page may check that the visitor performs some activity, such as moving the mouse or clicking on links, or that the browser possesses the idiosyncratic properties of commonly-used modern browsers, rather than being a simple emulator. If any of these checks are not satisfied, the malicious web page will refrain from launching the attack, and, as a consequence, will be incorrectly classified as benign, thus evading detection.

Two approaches have been recently proposed that can help mitigating this problem. Proponents of the "multi-faceted" approach suggest using a combination of different detection systems to classify web pages, with the assumption that it is more difficult for an attacker to develop an evasion technique that works against all the systems. Unfortunately, setting up and maintaining different systems incurs significant operational and management cost.

The second approach consists of using "multi-execution techniques" to explore multiple, alternative paths in the code of the malicious web page. The assumption in this case is that, by forcing the execution through alternative branches, the detector may be able to bypass the evasion checks implemented in a page and reveal its full behavior. Unfortunately, for this approach to be feasible, the multi-path execution is limited to "interesting" control-flow branches, i.e., those that depend on the value of specific objects (e.g., the `navigator.plugins` array that list available plugins). Evasive code that relies on different, unexpected checking techniques will not trigger the multi-execution machinery and will thus remain undetected. In addition, malicious code could repeatedly trigger the multi-path exploration to make this analysis computationally infeasible.

In this paper, we take a different approach to automatically identify evasion attempts in drive-by-download code. Our approach, called *Revolver*, retains the ability of detecting generic evasion techniques (both known and unknown) without incurring the overhead of multiple detection systems. In particular, our approach is based on a simple observation: two scripts that are *similar* should be classified in the same way by a drive-by-download detector; that is, they should both be flagged as benign or both as malicious. Different detection classifications (all other things being equal) can be attributed to the differences in the scripts. Since these differences cause a malicious script to be classified as benign, they may correspond to implementations of evasion attempts.

Notice that this approach requires neither the predetermined definition of a set of "interesting" code features (as in multi-execution tools) nor the availability of classification results from multiple detectors (as in multi-faceted approaches).

In practice, given a piece of JavaScript code, *Revolver* efficiently identifies scripts that are *similar* to that code, and automatically classifies the differences between two scripts that it has determined to be similar. In particular, *Revolver* focuses first on identifying syntactic-level differences in similar scripts (e.g., insertion, removal, or substitution of snippets of code) and then on explaining the semantics of such differences (their effect on the page execution).

There are several challenges that *Revolver* needs to address to apply this approach in practice. First, typical drive-by-download web pages serve malicious code that is heavily obfuscated. The code may be mutated from one visit of the page to another by using simple polymorphic techniques, e.g., by randomly renaming variables and functions names. Polymorphism creates a multitude of differences in two pieces of code. For a superficial analysis, two functionally and structurally identical pieces of code will appear as very different. In addition, malicious code may be produced on-the-fly, by dynamically generating and executing new code (through JavaScript and browser DOM constructs such as the `eval()` and `setTimeout()` functions). Dynamic code generation poses a problem of coverage, that is, not all JavaScript code may be readily available to the analyzer. Therefore, a naive approach that attempts to directly compare two malicious scripts would be easily thwarted by these obfuscation techniques and would fail to detect their similarities. Instead, *Revolver* monitors the execution of JavaScript code in a web page so that it can analyze both the scripts that are statically present in the page and those that are dynamically generated. In addition, to overcome polymorphic mutations of the code, *Revolver* performs its similarity matching by analyzing the Abstract Syntax Tree (AST) of code, thereby ignoring superficial changes to its source code.

Second, the analysis needs to scale. In fact, in a typical analysis of a web page, *Revolver* needs to compare several JavaScript scripts (more precisely, their ASTs) with a repository of millions of ASTs (potential matches) to identify similar ones. To make this similarity matching computation efficient, we use a number of machine learning techniques, such as dimensionality reduction and clustering algorithms.

Finally, not all code changes are security-relevant. For example, a change in a portion of the code that is never executed is less interesting than one that causes a difference in the runtime behavior of the script. In particular, we are interested in identifying code changes that cause detection tools to misclassify a malicious script as benign. To identify such evasive code changes, *Revolver* focuses on modifications that introduce control-flow changes in the program. These changes may indicate that the modified program checks whether it is being analyzed by a detector tool (rather than an unsuspecting visitor) and exhibits a different behavior depending on the result of this check.

Our main goal with *Revolver* is to identify code changes designed to evade drive-by-download detectors: this knowledge can be used to improve detection tools and to increase their detection rate. We also can leverage *Revolver* to identify benign scripts (e.g., well-known libraries) that have been injected with malicious code, and, thus, display malicious behavior. Finally, with the use of *Revolver*we can demonstrate several use cases by analyzing differences between similar pieces of malicious code that can provide interesting insights into the techniques and methods followed by attackers.

The contributions of our work are the following:

1) **Code similarity detection**: We introduce techniques to efficiently identify JavaScript code snippets that are similar to each other. Our tool is resilient to obfuscation techniques, such as polymorphism and dynamic code generation, and also pinpoints the precise differences (changes in their ASTs) between two different versions of similar scripts.
2) **Detection of evasive code**: We automatically classify differences between two similar scripts to highlight their purpose and effect on the executed code. In particular, we discovered several techniques that attackers use to evade existing detection tools.

II. Conclusions

In this paper, we have introduced *Revolver*, a novel approach and tool for detecting malicious JavaScript code that attempts to evade detection. *Revolver*'s approach is based on identifying scripts that are similar yet are classified differently by malicious code detectors, e.g., a honeyclient. *Revolver* analyzes differences in such scripts to isolate evasion attempts.

References

[1] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: A Fast Filter for the Large-scale Detection of Malicious Web Pages. In *Proceedings of the International World Wide Web Conference (WWW)*, 2011.

[2] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. In *Proceedings of the International World Wide Web Conference (WWW)*, 2010.

[3] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna. Escape from Monkey Island: Evading High-Interaction Honeyclients. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2011.

Work-in-Progress: Assembly Code Generation for Arbitrary Rotations

Daniel Kudrow, Kenny Bier, Oana Theogarajan
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, USA
dkudrow@cs.ucsb.edu

Fred Chong, Diana Franklin
Department of Copmuter Science
University of California, Santa Barbara
Santa Barbara, USA
chong@cs.ucsb.edu, franklin@cs.ucsb.edu

***Abstract*—Analysis of existing quantum algorithms has revealed a significant need for arbitrary rotations. These rotations are not always known at compile-time and even when they are, static compilation can result in a terabyte of generated assembly code.**

We describe a study in which we evaluate the design space defined by required precision, approximation method, and quantum device technology. The precision required and approximation method affect the length of the generated code sequence as well as the time required to generate the code. The quantum technology determines the speed at which dynamic code generation must be accomplished to avoid slowdown. We also evaluate the trade-offs between static and dynamic code generation.

***Index Terms*—quantum computing; Solovay-Kitaev;**

I. Introduction

A quantum computer is a model of computation that utilizes features of quantum mechanics to manipulate information. The qubit (quantum bit) is the basic unit of data in a quantum computer and represents information stored in a quantum state. The Bloch Sphere of a qubit is a useful visual aid in understanding this phenomenon.

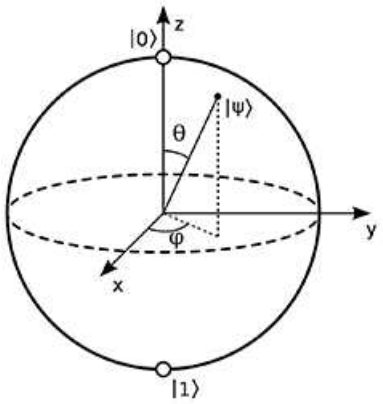


Fig. 1. A Bloch Sphere for 1 qubit

The state of a qubit is depicted as a point on the surface of the Sphere. The classical states 1 and 0 are located at the poles of the Sphere. All other points represent a *superposition* of these two states. Algorithms that exploit this feature of a quantum computer have the potential to exponentially increase their speed because n qubits represent a superposition of 2^n classical states and a quantum computer can act on all of these states simultaneously.

A. Arbitrary Rotations

The ability to shift the phase of a qubit by an arbitrary angle has proven to be fundamental to many quantum algorithms. Shifting the phase of a qubit corresponds to a rotation about the Z-axis of that qubit's Bloch Sphere and is accomplished by applying the $R_z(\theta)$ quantum gate:

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{bmatrix} \quad (1)$$

.

The difficulty in implementing rotations for arbitrary angles stems from the fact that $R_z(\theta)$ represents a continuous set of gates (for all possible angles). Fault-tolerant constructions are generally available only for a small set of gates[1], so a fault-tolerant $R_z(\theta)$ must be approximated by a sequence of gates from such a set. In designing a general purpose quantum computer it is important to consider how and when such sequences will be generated to optimize performance.

II. Approximation Accuracy

Accuracy is the most important factor to consider in sequence generation. The accuracy of an approximation is the rotational distance between it and the original gate. This is formally defined as the projected trace distance between the two matrices.

A. Existing Algorithms

The accuracy required by an algorithm can be approximated by examining the number of qubits measured after the application of a gate. A measurement on a register of n qubits must be accurate to 2^{-n}. Rotation errors propagate linearly so r rotations of accuracy d will generate error $r \times d$. Thus the minimum accuracy per gate for succesful execution of the algorithm is $d < \frac{2^{-n}}{r}$.

We studied the algorithms in the IARPA QCS algorithm suite that include arbitrary rotations to determine what levels of accuracy were required and detail results below.

TABLE I
ROTATIONS IN EXISTING ALGORITHMS

Algorithm	Total Rotations	Required Precision
Boolean Formula	53,298	2.4e-62
Linear Systems	1,794	1.2e-23
Binary Welded Tree	2,080	2.9e-8
Shortest Vector	unknown	unknown
Ground State Estimation	2.5e14	9.7e-50

B. Solovay-Kitaev

We have chosen to approximate rotations using the Solovay-Kitaev algorithm[2] with H, T, H^{-1}, T^{-1} as our universal set of gates. The implementation we are using, written in C++ by Chris Dawson and Michael Neilsen, is limited by the precision of the system floating point data type. We modified the original algorithm, written by Chris Dawson and Michael Neilsen to incorporate arbitrary precision floating point numbers using the CLN Library for C++. Whereas the original algorithm was limited to an accuracy of around 1×10^{-11}, our altered version accommodates arbitrarily precise rotations. We demonstrate our modification in the plot below. We ran Solovay-Kitaev to a recursion of depth six on the rotations $\theta = \frac{\pi}{2^k}$ for a range of k values. It is clear that for values of $k \geq 30$ the original algorithm is no longer useful.

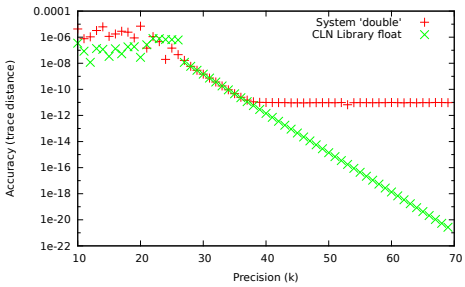


Fig. 2. Performance of arbitrary precision Solovay-Kitaev algorithm

III. COMPILATION TECHNIQUES

Aside from a method for generating sequences, it is important to consider when this process will occur. There are advantages to generating sequences both during compilation as well as at run-time.

A. Compile-Time Sequence Generation

The simplest approach to approximating rotations is to generate sequences at run-time. The advantage of this approach is that there is no need to dynamically generate sequences at run-time for each execution. The cost for arbitrary rotations is paid once and can then be neglected. The primary drawback to this approach is the size of the generated code which will included the full sequence for each rotation. Furthermore this approach cannot be used on rotations that are not known at compile-time.

B. Run-Time Sequence Generation

When compiling algorithms in which rotations are not known at compile-time, sequences will have to be generated at run-time. The important consideration in this approach is the time taken to approximate a rotation. The time taken to generate sequences on the fly will not only increase execution time but, in the worse case, can allow qubits to decohere, thus breaking the algorithm.

C. Hybrid Approach

The likeliest implementation of a quantum compiler will probably combine the above two approaches. One implementation of this strategy would be to generate a basis of sequences during compilation, such as rotations of $\frac{pi}{2^k}$ for $k = 1, 2, ..., n$. Then these sequences can be concatenated to form an approximation of the form $\sum_{k=0}^{p} \frac{pi}{2^k}$ with precision $\frac{pi}{2^p}$.

IV. CONCLUSION

It is clear that arbitrary rotation gates are going to be crucial to the implementation of a general purpose quantum computer. As the title indicates, this study is by no means exhaustive. There remain methods of sequence generation and compilation strategies that will need to be thoroughly examined.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computing and Quantum Information*, New York, USA: Cambridge University Press, 2000.
[2] C. M. Dawson and M. A. Nielsen, The Solovay-Kitaev Algorithm, *Quantum Computation and Information* 0:000-000, 2005.

Do you feel lucky? A large-scale analysis of risk-rewards trade-offs in cyber security

Yan Shoshitaishvili
UC Santa Barbara
yans@cs.ucsb.edu

Adam Doupe
UC Santa Barbara
adoupe@cs.ucsb.edu

Luca Invernizzi
UC Santa Barbara
invernizzi@cs.ucsb.edu

Giovanni Vigna
UC Santa Barbara
vigna@cs.ucsb.edu

I. Abstract

A cyber-criminal's profit is determined by the balance of the risks and rewards of her every action. For example, she might tune a spamming bot's email-sending rate to achieve a good throughput, with an acceptable risk of being detected. A large-scale study on how cyber-criminals deal with these risk-reward trade-offs is challenging, as it requires the participation of many volunteers that are knowledgeable in this field.

Computer security competitions provide a great opportunity both to educate students and to study realistic cyber-security scenarios in a controlled environment. Looking to model the risk-reward trade-offs seen in real cyber-security incidents, we designed and hosted a novel format for a Capture the Flag cyber-security contest. The competition, which took place in December 2011, was the largest educational live security exercise ever performed, involving 89 teams comprising over 1,000 students across the globe. In this paper, we describe the intuition, intent, and design of the contest.

II. Introduction

Computer security incidents commonly display a risk-reward trade-off. Examples of this in the wild are plentiful: a spear-phishing campaign might target an increased amount of users at the cost of an increased risk of being reported, a backdoored system could send data faster but risk detection by an IDS, or a bot could spam at a higher frequency and risk being blacklisted. However, reproducing these scenarios in a controlled environment is a complicated problem. Continuing from last-year's success [**?**] in generating a dataset from a cyber-security competition, we designed and organized a new one on this theme. The competition, which took place at the end of 2011, saw academic teams from around the world understanding and adapting to this new scenario.

As computer security becomes an increasingly important issue for modern corporations and governments, the question of training the next generation of security professionals rises in significance. One solution to this, known in the community as a *Capture the Flag* (CTF) competition.

A CTF contest is a cyber-warfare exercise consisting of a set of goals, with the participating teams in competition for the completion of these goals. Completion of a goal by a participating team generally results in a reward of points, and the team with the most points at the end of the competition wins. Depending on the CTF, the completion of these goals might require skills in many areas of computer security.

This year, the UCSB iCTF competition was attended by 89 teams, comprising over 1,000 students. All teams were vetted to ensure that they are academic teams, participating with the guidance of a faculty member at an accredited university. These faculty mentors are expected to help ensure ethical behavior by the participants and to provide a reliable point of contact during the competition. Of course, we scaled the difficulty of the challenges of our competition accordingly, creating challenges for players of all skill levels.

In this paper, we describe the design and implementation of a novel, large-scale Internet CTF contest, attended by 89 teams from 18 countries, totalling over 1,000 students.

III. Competition Story

We chose a theme of illegal money laundering for our competition. This activity is modeled after cyber-criminal money laundering operations and implements several risk-rewards trade-offs. Furthermore, a money laundering theme provides a perfect setting for risk-reward analysis, as the trade-offs are very intuitively understood.

IV. Competition Overview

The general idea behind our competition was the conversion ("laundering") of money, obtained by solving challenges from the challenge-board, to points by utilizing data captured from an exploited service of an opponent team. Successful conversion of money to points depended on a number of factors, calculated together as the "risk function". The factors of the risk function would change every *tick*. Each tick lasted two minutes, fast enough to force teams to automate the choice of services through which they would launder money.

Several intuitions brought us to the idea for this year's contest. The most important consideration was the development of a competition to properly model the risk-reward trade-offs described in Section II. However, an additional consideration was the assurance of a fair and exciting competition. In many CTF competitions, one team is able to carry the entire competition by utilizing experience in a specific field, completely ignoring whole portions of a competition. We wanted all aspects of this competition to be important, so we devised a competition format that would put heavy emphasis on both team-vs-team competition and on the challenge-board. This was the reasoning behind implementing the money-to-points flow of the competition.

In a similar vein, we wanted to prevent a team from skyrocketing in points and creating an unassailable lead, since this can have the effect of discouraging other players. We wanted to promote a format where teams that pulled ahead had to fight to maintain their position while at the same time providing a challenge for the other teams trying to catch up. These motivations helped shape the risk function itself for the conversion of money into points, and helped us arrive at a design where money-hoarding was possible.

V. Competition Design

Our CTF required a carefully-planned infrastructure, which was designed and deployed for maximum network performance and uptime.

Network To minimize the potential damage from a security compromise, we created a split network approach, consisting of three levels of trust (internal, DMZ, and untrusted).

Player VMs A virtual machine, containing the services, was released in an encrypted state and the encryption key was released at the start of the competition.

Services A service is a network application, run by the participants, which carries out some defined functionality but contains an intended security vulnerability. Each team must exploit such vulnerabilities in the services of other teams, while protecting their own services from exploitation.

To check for the availability of these services, we created a scorebot that would connect to each service and run through the standard usage scenarios for it. Our scorebot was a program checked the status of each service at random intervals. The percentage of time that a team's service was up was used as a scaling factor in the risk function for laundering attempts by that team.

Flags Each service has access to a "flag," which the attacker must capture by exploiting the services (hence the name Capture the Flag). To require teams to continue carrying out attacks, these flags are regularly updated. This allows the organizers to track the defensive capabilities of the teams as they adapt and block existing exploits, and the ability of the attackers to regain access.

Our scorebot would update each service's flag as a normal part of a service's operation. For example, our scorebot might act as the user of a messaging service, setting the flag as the user's password. To recover the flag, an attacker would have to compromise the service to the point of being able to recover user passwords.

Challenges Aside from exploiting services, our competition required teams to solve challenges (in a form of a file or web page to analyze) to earn in-game money. Successful analysis of the challenge would yield the "key" in the form of a string, and submission of this value into the web form would net the submitting team an amount of money depending on the difficulty of the challenge.

Risk The overall risk is the probability that a team's laundering attempt succeeds. There are four factors that influence a team's laundering success: R, the risk of the service; M, the amount of money the team is trying to launder; N, the total amount of money the team has laundered through that team; and Q, the total amount of money the team has laundered through that service. In this way, we encouraged the teams to switch the team that they laundered through and to switch the services that they used to launder.

The factors of the risk function changed every two minutes. We called the times that these values changed a tick. We generated these values based on the status of four internal (not provided to the teams) situational awareness missions. We used the same representation of these missions as in the 2010 iCTF [**?**]—petri nets. However, in this competition, the states of the petri nets influenced the risk values.

Laundering The goal of the CTF was to earn money by solving challenges and convert it into points by *laundering* them through exploited services. When a team exploited a service, and had some money to convert, they would submit the captured flag to the submission server and choose an amount of money to launder. This submission server would calculate the risk (as described above) and roll a virtual die to determine if the attempt was successful. If the roll was successful, the submission server would calculate the appropriate gain in points by the team involved, and adjust points accordingly. However, if the laundering failed, the team would simply lose its money. In either case, the submission server would mark the exploited teams' exploited service as being compromised.

The Scoreboard During a CTF, the scoreboard always attracts more load than any other part of the infrastructure. For this reason, we took a very scalable approach to hosting the scoreboard for our competition. The board was re-generated once per tick, and contained no server-side dynamic content. Furthermore, it was deployed on a Eucalyptus [**?**] cloud, just in case the servers could not keep up. This turned out to be necessary, as the load that the scoreboard experienced necessitated the engagement of every available node in our cloud.

VI. Results

Our design for the competition proved to be a valid one, providing intense team-vs-team competition until the very end of the contest. Statistical analysis of the performance of the top 10 teams found that the single most important factor in their placing was their *Conversion Ratio*, which we defined as the percentage of money that was successfully converted into points. The correlation between points and conversion ration was 0.81, while the correlation between points and the amount earned was only 0.71. This result was reflected in the fact that the top team did not have the most earned money by a fairly wide margin. When analyzing all teams, instead of just the top 10, however, earned money correlates more strongly with their points than does conversion ratio.

References

[1] N. Childers, B. Boe, L. Cavallaro, L. Cavedon, M. Cova, M. Egele, and G. Vigna. Organizing large scale hacking competitions. *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 132–152, 2010.

[2] A. Doupé, M. Egele, B. Caillat, G. Stringhini, G. Yakin, A. Zand, L. Cavedon, and G. Vigna. Hit'em where it hurts: a live security exercise on cyber situational awareness. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 51–61. ACM, 2011.

[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

Blacksheep: some dumps are dirtier than others

Antonio Bianchi
UC Santa Barbara
antoniob@cs.ucsb.edu

Yan Shoshitaishvili
UC Santa Barbara
yans@cs.ucsb.edu

Chris Kruegel
UC Santa Barbara
chris@cs.ucsb.edu

Giovanni Vigna
UC Santa Barbara
vigna@cs.ucsb.edu

***Abstract*—The lucrative rewards of security penetrations into large organizations have motivated the development and use of many sophisticated rootkit techniques to maintain an attacker's presence on a compromised system. Due to the evasive nature of such infections, detecting these rootkit infestations is a problem facing modern organizations. While many approaches to this problem have been proposed, various drawbacks that range from signature generation issues, to coverage, to performance, prevent these approaches from being ideal solutions.**

In this paper, we present *Blacksheep*, a distributed system for detecting a rootkit infestation among groups of similar machines. This approach was motivated by the homogeneous natures of many corporate networks. Taking advantage of the similarity amongst the machines that it analyses, *Blacksheep* is able to efficiently and effectively detect both existing and new infestations by comparing the memory dumps collected from each host.

I. Introduction

Over the past several years, computer security has taken the center stage, as several high-profile organizations have suffered costly intrusions. Oftentimes, as in the case of the 2011 RSA compromise, such intrusions begin as a foothold on a single infected machine and spread out from that foothold to infect a larger portion of the enterprise. In the case of the 2010 Stuxnet attack on Irani nuclear reactors, this infection took the form of a kernel-based rootkit.

Rootkits are pieces of software designed to stealthily modify the behavior of an operating system in order to achieve malicious goals, such as hiding user space objects (e.g., processes, files, and network connections), logging user keystrokes, disabling security software, and installing backdoors for persistent access. Although several detection and prevention techniques have been developed and deployed, all have considerable drawbacks, and as a result, rootkits remain a security threat: According to recent estimates, the percentage of rootkits among all anti-virus detections is in the range of 7-10% [1], [3].

The goal of our work is to detect kernel rootkits, a broad class of rootkits that operate by modifying kernel code or kernel data structures. We focus on the Windows operating system, since it is both the most widespread and the most targeted platform. However, most of the concepts and techniques used are applicable to any operating system.

The observation that motivates our approach to the detection of rootkits is the fact that modern organizations rely on large networks of computers to accomplish their daily workflows. In order to simplify maintenance, upgrades, and replacement of their computers, organizations tend to utilize a standard set of software and settings for the configuration of these machines. We believe that by leveraging the similarities between these computers, malware can be detected with higher accuracy and without the limitations of modern malware detection techniques.

Therefore, we propose a novel technique for detecting kernel rootkits, based on the analysis of physical memory dumps taken from running operating systems. In our approach, a set of memory dumps from a population of computers with identical (or similar) hardware and software configurations are taken. These dumps are then compared with each other to find groups of machines that are similar. Finally, these groups are further analyzed to identify the kernel modifications introduced by a potential rootkit infection. In particular, we look for outliers that are different than the rest. Our insight is that these differences are an indication of a malware infection.

We implemented our approach in a tool, called *Blacksheep*, and validated it by analyzing memory dumps taken from two sets of computers. *Blacksheep* has several advantages over the state of the art. First of all, *Blacksheep* can detect stealthy rootkit infection techniques, such as data-only modifications of kernel memory. Additionally, *Blacksheep* does not need to be configured to detect specific modifications, because it relies on the identification of anomalies among a group of similar hosts. This means that *Blacksheep* does not use or rely on signatures, and can detect 0-days as effectively as it can detect long-known threats.

Since *Blacksheep* bases its analysis off of a crowd of similarly-configured machines, the system can be used on groups of machines in which some instances are already infected with malware. As long as a viable memory dump can be obtained, and as long as the majority of the machines comprising the crowd are not compromised, *Blacksheep* will be able to identify infections by comparing the memory dumps of the involved machines. In contrast, prior tools that utilize comparative techniques on data from a single machine cannot be safely deployed onto infected computers, since they would then have no safe baseline against which to compare.

II. Approach

Blacksheep is designed to detect rootkit infestations in kernel memory. *Blacksheep*'s design is motivated by the realization that, regardless of how much a rootkit tries to hide itself, it must still be accessible by the operating system in order to be executed. This concept is known as the Rootkit Paradox [2]. Additionally, even if a rootkit manages to hide its code from the operating system, the data modifications it makes can still be detected.

Since we are comparing kernel memory snapshots, an understanding of this memory space is required. The Win-

dows kernel consists of many *modules*, which are PE files containing kernel code and data. Modules can be operating system components (e.g., kernel32.dll) or hardware drivers (e.g., nvstor32.sys), and we use these terms interchangeably. The module and driver files are loaded into kernel memory in much the same way as dynamically linked libraries (DLLs) are loaded into user-space programs, and make up the functionality of the kernel. Similar to Windows DLLs, kernel modules contain both code and data segments. These segments require separate approaches in their comparisons, and *Blacksheep* treats them separately.

In summary, *Blacksheep* performs the following four types of analyses.

Configuration comparison. Some rootkits come in the form of a kernel module that is loaded into the system. To identify such changes, *Blacksheep* does a "configuration comparison," comparing loaded modules between two memory dumps. This allows the system to detect additional (and potential malicious) components that are introduced into the kernel.

Code comparison. Most rootkits directly overwrite or augment existing code in kernel space with malicious content so that they can perform subversive tasks (such as hiding resources) when this code is executed. Thus, a difference in kernel code between machines that should be otherwise identical can be a good indicator that a rootkit may be present. However, due to the possibility of benign differences resulting from, among other causes, code relocation and anti-virus defense techniques, a detected difference might not necessarily mean that the machine is infected. *Blacksheep* can filter out benign differences and focus on suspicious code differences.

Data comparison. Detecting differences in kernel code alone is not enough to detect the presence of rootkits with high accuracy. For example, certain rootkits are able to subvert system functionality without performing any modifications to code running on the system, and, instead, they change kernel data structures to avoid detection through code comparison. Because of the threat of such rootkits, we compare kernel data between machines.

Comparing such data between two different machines is a non-trivial task, and constitutes a large portion of *Blacksheep*'s contribution. For statically allocated data segments (i.e., those segments that are defined in and loaded from the PE file), the main challenge is handling relocation. However, dynamically allocated memory provides a more substantial challenge. This data oftentimes contains many layers of data structures linking to each other, which must be navigated in order to ensure good coverage. *Blacksheep* uses an heuristic approach to detect data structures. Then, it recursively compares them among memory dumps, focusing on those differences that can be caused by a rootkit infection.

Entry point comparison. Additionally, rootkits might subvert basic interfaces to the Windows kernel in order to carry out their tasks. This includes the Windows kernel SSDT, driver IRP communication channels, and certain hardware registers in the x86 architecture. *Blacksheep* is able to compare such *kernel entry points* by processing the machines' dumps of physical memory.

Clustering and detection. After comparing each pair of memory dumps, *Blacksheep* places them into a hierarchy of clusters. The larger clusters are then assumed to contain the clean dumps, and the smaller clusters are labeled as suspicious. The assumption is that only a small fraction of the hosts are infected, and these hosts stand out as outliers when compared to the other machines in the crowd.

III. Evalutaion

We evaluated *Blacksheep* on two sets of memory dumps. The first was acquired from a set of Windows 7 virtual machines using QEMU VM introspection, the second set of memory dumps was acquired using a memory acquisition driver developed by us. We tested these configurations against a range of publicly available rootkits. In particular, we used the well-known Mebroot, Stuxnet, Rustock, and Blackenergy rootkits, three rootkits in the TDSS family (tdss, tdl3, and tdl4), and the r2d2 Trojan. Unfortunately, 3 out of 5 rootkits do not function properly on Windows 7, so the range of tested rootkits is smaller for the first data set (Windows 7 - QEMU Introspection).

Windows 7 - QEMU Introspection. We tested *Blacksheep* against a set of 40 memory dumps taken through QEMU VM introspection. Within the set, 20 of the dumps were clean, and 20 were infected with rootkits, with 4 machines infected with each of 5 rootkits. *Blacksheep* achieves a true positive rate of 100% and a false positive rate of 0%.

Windows XP - Driver-acquired Memory. *Blacksheep* was also tested in detecting rootkits on Windows XP. Again, 10 clean dumps were clustered, this time together with 8 rootkits. *Blacksheep* produces 75% true positives and 5.5% false positives.

References

[1] A. Kapoor and R. Mathur. Predicting the future of stealth attacks. *Virus Bulletin conference*, Oct. 2011.

[2] J. D. Kornblum. Exploiting the rootkit paradox with windows memory analysis. *International Journal of Digital Evidence*, 2006.

[3] R. Treit. Some observations on rootkits, Jan. 2010. *http://blogs.technet.com/b/mmpc/archive/2010/01/07/some-observations-on-rootkits.aspx*.

Directed Social Queries

Saiph Savage[1] Angus Forbes[1] Rodrigo Savage[2] Norma Elva Chávez[2] Tobias Höllerer[1]

[1]University of California, Santa Barbara
{saiph@cs, angus.forbes@cs, holl@cs}.ucsb.edu

[2]Universidad Nacional Autónoma de México
{rodrigosavage@comunidad, norma@fi-b}.unam.mx

***Abstract*—The friend list of many social network users can be very large. This creates challenges when users seek to direct their social interactions to friends that share a particular interest. We present a self-organizing online tool that by incorporating ideas from user modeling and data visualization allows a person to quickly identify which friends best match a social query, enabling precise and efficient directed social interactions. To cover the different modalities in which our tool might be used, we introduce two different interactive visualizations. One view enables a human-in-the-loop approach for result analysis and verification, and, in a second view, location, social affiliations and "personality" data is incorporated, allowing the user to quickly consider different social and spatial factors when directing social queries. We report on a qualitative analysis, which indicates that transparency leads to an increased effectiveness of the system. This work contributes a novel method for exploring online friends.**

I. Introduction

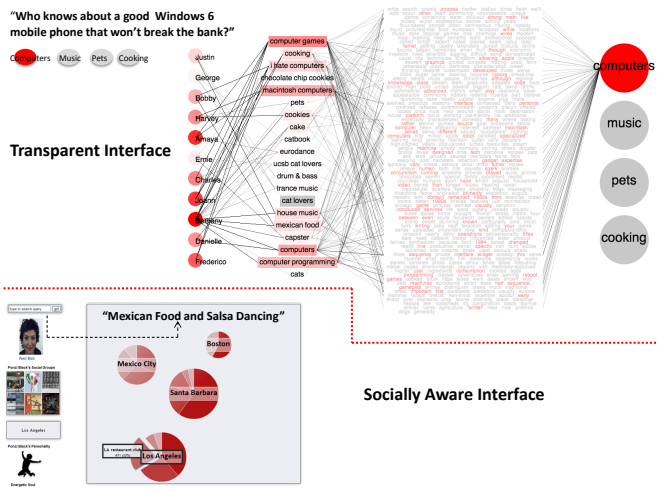


Fig. 1. Screenshot of our two interactive interfaces. Top shows the transparent interface, with the friends most related to the social query: "Who knows about a good Windows 6 phone that won't break the bank?". To the right of the highly correlated friends' names, we see their associated likes, their meanings and the mapping to the shared interests. The four shared interests shown, are the ones the system automatically found for the friends of a particular user. The user can explore and analyze all the data the system used to recommend that particular set of friends. The socially aware interface is shown below. This interface is using spatial and social variables to organize friends related to the social query: "Mexican Food and Salsa Dancing". In the left side of this interface, the user can analyze particular characteristics of a specific friend. Our system allows for various different types of social query formulation; a specific query format is not required.

An increasing number of individuals regularly use social networks as a platform from which to pose questions and engage in playful social interactions. The questions and social interactions are often directed to a subset of the user's friends who are potentially helpful in answering a particular question or who share a particular interest. While many social network sites already let users define collections of people in various ways, manually classifying friends into collections is time consuming, and even if users were to finish this exhaustive categorization task, the predefined lists might not cover all of the user's intended social interactions, especially as social groupings dynamically shift. For example, a user might be hosting an event related to a particular theme and seeking to invite only the friends that are interested in that theme. In this case, the user's predefined lists might be too coarse or otherwise inappropriate for this particular task. Therefore, a system which could automatically find friends given a user's social query would be beneficial. Bernstein et.al [2] addressed this problem, by presenting a system which recommended the friends a user should share particular web content to. The work of Amershi et. al [1] presented a system that helped people create custom, on-demand groups in online social networks. Despite their novelty, these systems did not offer much transparency to the inner workings of their approach, and this could impact user experience. Furthermore, while [1]'s work let the user filter group members based on particular attributes, their system did not directly allow a user to target individuals based on a particular social query. Additionally, both of these studies paid little attention to the overall personality of each of the user's friends, which is an attribute that has been shown to play an important role in online and physical social interactions [3]. These insights lead us to create a transparent online tool that effectively matches people to social queries, and also offers information on particular social and spatial factors related to these individuals, such as personality traits, and social-spatial affiliations. Our system has 3 parts: a machine learning part that infers the interests shared by a user's friends and finds friends relevant to a social query, a transparent verification interface, and a socially-aware interface.

Modeling Shared Interests: A friend's interests are determined through his/her Facebook *likes*. Given that the textual information describing a *like* can be very sparse and thus difficult for a machine to interpret, our system retrieves additional information that can provide a broader semantic description, and expose the potential meaning of the *like*. The additional data is obtained by using the name of each of the *likes* as a search phrase in a crowd-sourced knowledge base (Wikipedia) and collecting all the textual data that relates to the *like*. Each friend is then linked to a bag of words representing *likes* and their meanings. A user modeling approach similar to that of [4] is used to find from this data, how related each friend is to a set of automatically derived "shared interests", that broadly define the different tastes of a user's friends. A "shared interest" is defined as a set of words that frequently co-occur together in the bag of words of all of the user's friends. Figure 1 shows the different "shared interests" the system discovered for the friends of one particular user. When the user types a social query, this query is also modeled in terms of these shared interests, and the K friends who are the most relevant to the query, are recommended to the user. For visualization

purposes K=11 was chosen in this case.

Transparent Verification Interface: This component offers a visualization of the friends who are most correlated to a particular social query. It also links and presents the data utilized in recommending that particular set of friends, thus allowing a user to verify if a recommendation is indeed appropriate. Figure 1 shows a screenshot of the interface when the user typed the social query: "Who knows of a good Windows 6 phone that won't break the bank?". We ran a series of in-depth cognitive walkthroughs with a small number of subjects to solicit feedback about the basic design as well as to identify how effective this type of transparency might be for identifying appropriate friends for directed queries. All of our subjects were able to navigate the represented information within a few seconds and only minimal instruction. For the most part, the subjects had positive responses to the visualization, and noted that it aided them in verifying and correcting friend lists. As a test case, we included two friends who were purposefully correlated with shared interests incorrectly. In order to see if users would notice that some correlations were incorrect. Without exception, users independently noticed that something was awry without any indication from the experimentors. Our motivation in providing this example was to show that even a simple visualization of the underlying data was sufficient to allow a subject to confirm or refute a classification. The simple visual cues in these instances were sufficient to indicate issues with the model and to cause users to investigate their cause.

Socially-Aware Interface: Although our transparent interface is effective at weeding out problematic correlations, it does not consider social contexts or spatial temporal constraints, which are known to affect a user's preferences and decisions [5]. We extended our system to address potential real-world scenarios. To this end, we organize highly-correlated friends in terms of their geographical location and their inclusion in particular Facebook groups. Through visualizing friends in this way, a user can quickly determine which of the users are appropriate for a particular social query. This is especially true if the query may only have local relevance. There are also several different social constraints that can play a role in the user's desire to direct a social query to a particular friend. For example, a user organizing an event to prepare for a competition might not wish to invite rivals from the other team. We also considered that a friend's overall personality might play a role in the user's desire to include the friend in the directed social query. A friend's personality is determined by calculating the similarity index that the friend's *likes* and their associated textual meaning present with the adjectives from Thayer's Activation-Deactivation Adjective Check List [6]. A friend's personality is classified as either energetic calm, energetic tense, tired calm or tired tense. For entertainment purposes, the following categories were used in our interface: "energetic soul", "high strung", "sleepy soul", and "anxiously drowsy".
It is important to note, that the friend features used here were for showing the potential machine learning and visualization have in aiding social decisions; they are not exhaustive.

II. Conclusion

This paper introduces a novel system for modeling and recommending social network users based on content and social factors. To mitigate potential machine learning modeling errors, we introduce two interactive visualizations that allow for result analysis and verification. A user study evaluating our system is forthcoming. We believe our system offers a novel clear fast way for directed social querying and allows a user to explore and learn about her friends in a new way.

III. ACKNOWLEDGMENTS

This work was partially supported by CONACYT-UCMEXUS & NSF grant IIS-1058132. Special thanks to Janet L. Kayfetz for her motivation.

References

[1] Amershi, S., Fogarty, J., Weld, D.S. ReGroup: Interactive Machine Learning for On-Demand Group Creation in Social Networks.Proc. CHI '12, ACM Press(2012)

[2] Bernstein,M., Marcus,A., Karger, D., Miller R. Enhancing directed content sharing on the web. Proc. CHI '10, ACM Press (2010)

[3] Caropreso, E., Chen, S.J. Effects of Personality on Small Group Communication and Task Engagement, Proc.of World Conference on E-Learning, AACE (2005)

[4] Dietz,L.,Gamari B.,Guiver J.,Snelson E.,Herbrich E. De-Layering Social Networks by Shared Tastes of Friendships. ICWSM The AAAI Press (2012)

[5] Savage,N.S,Baranski, M.,Chavez,N.E.,Hollerer T. I'm feeling LoCo: A Location Based Context Aware Recommendation System. Proc. 8th International Symposium on Location-Based Services, Lecture Notes in Geoinformation and Cartography,Springer (2011)

[6] Thayer, R.E., Activation states as assessed by verbal report and four psychophysiological variables. Psychophysiology 7, 1 (1970), 86-94. (1970)

Delay Injection for Service Dependency Detection

Ali Zand*, Christopher Kruegel*, Richard Kemmerer* and Giovanni Vigna*
*Computer Security Lab
University of California Santa Barbara,
Santa Barbara, California 93106-5110
Email(s): {zand,chris,kemm,vigna}@cs.ucsb.edu

***Abstract*—Detecting dependencies among network services has been well-studied in previous research. Unfortunately, previous work suffers from false positives and applicability issues. In this paper, we provide a new approach for detecting dependencies among services. We present a traffic watermarking approach with arbitrarily low false positive and easy applicability.**

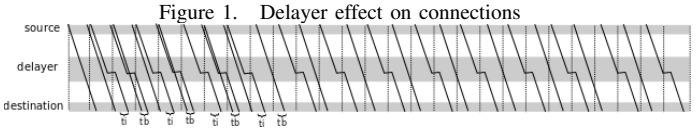
I. Introduction

With every day advancement of computer networking technology, network services have taken more important roles in every day life. We depend on these network services for many of our daily needs. As with great role comes great complexity, these network services are normally implemented as composite services composed of multiple simpler underlying services, as a way to manage the complexity. This approach enables designers to reuse standard underlying services to build complex customized services. For example, a webmail service is usually implemented using several simple services such as web service, mail service, DNS service, et cetera. As the services become more composite and more complex, they need more protection, as there are more things that can go wrong and make the whole service to fail. One often needs to know the components of a composite service to be able to protect it. Unfortunately, these implementation and dependency details are often missing or incomplete in current computer networks.

Previous work on service dependency detection can be divided into active [1] and passive [2] approaches. Passive approaches generally do not generate any additional traffic and only observe the existing traffic to find correlated activity. Active approaches, on the other hand, make their own changes into the traffic. Passive approaches generally suffer from higher false positive problem. The reason for this problem is that correlation does not imply causation. In other words, when two services are detected as correlated to each other it does not necessarily mean that they depend on each other or, even if it does, it does not show which service depends on the other. Active approaches suffer from applicability problems, as they introduce more load into the network and they are usually application dependent. Application dependent approaches cannot be used for detecting dependencies between unknown types of services.

To reduce the false positive and tell the difference between correlation and causation, one needs to control one variable and observe the other. This is not possible with passive methods. Therefore, active approaches are needed to recognize the direction of the dependency problem. In this paper, we provide an active watermarking approach that is application independent.

Figure 1. Delayer effect on connections



II. Watermarking for Dependency Detection

In our approach, we assume that we have access to NetFlow records, or a similar data source, of the network traffic, and, we are also able to selectively delay packets.

We use traffic watermarking for detecting the dependency relations. More specifically, we perturb the timing of the connections destined to one service and observe whether the perturbation is propagated to the second service. Our approach is generic and application-independent, as we do not look into or change the packet contents. More specifically, to find what services depend on a given service S_1, we delay the first packet of each connection destined to service S_1. We expect to observe a similar delay in connections destined to services depending on S_1.

III. Induced Perturbation Model

Dependency detection problem can be modeled as follows. Assume that we want to test the dependency between services S_1 and S_2. We have already established that these services have correlated activity. The goal of the analysis is to determine if any of the two services depend on the other one. Service S_2 depends on service S_1 when a failure in service S_1 causes a failure in service S_2. Our hypothesis is that if service S_2 depends on service S_1, a delay d_1 in service S_1 should result in a similar delay $d_2 \approxeq d_1$ in service S_2.

A. Detection of the Injected Delay

Assume the following scenario. A connection C_1 to service S_1 is delayed. As a result, service S_1 will contact service S_2 through connection C_2 with a delay. The observer will see these two connections along with thousands of other connections and may not be able to map the two connections C_1 and C_2.

To make the perturbation observable to the observer, we create different perturbation patterns in different time windows and this will result in a similar perturbation pattern in the depending service.

To model the service activity, we divide the time into time windows of equal size ($w_1, w_2, \ldots, w_{2n}$, where $|w_i| = w$). We delay the requests destined to service S_1 for t_d in odd time windows $w_1, w_3, w_5, \ldots, w_{2n-1}$ and we do not delay them in even time windows. This process will create time windows with more than average requests (t_i idle time window) and time windows with less than average requests (t_b busy time window) on S_2, as shown in Figure 1. It is straight-forward to show that $t_i = t_b = t_d$.

Let's assume that the number of requests for service S_2 in different time windows (t_d) follows an unknown distribution $D_0 = D(\mu_0, \sigma_0)$, with mean and standard deviation equal to μ_0 and σ_0, respectively. Also, assume that ρ is the fraction of requests of S_2 that are caused by requests of S_1. It can be shown that the number of requests in the idle time windows and busy time windows follow $D_1 = D(\mu_0 \times (1-\rho), \sigma_0 \times (1-\rho))$ and $D_2 = D(\mu_0 \times (1+\rho), \sigma_0 \times (1+\rho))$, respectively. In other words, this watermarking results in consecutive periods of length t_d of distributions D_1 and D_2 separated from each other by periods of length $w - t_d$ of distribution D_0.

B. Statistical Inference

In general, we want to disprove that two services are independent.

X is the random variable for number of requests arriving for service S_2 in each time window of length t_d, when no delay is applied.

X_i and X_b are the random variables for number of requests arriving for service S_2 in each t_i and t_b time windows, respectively.

μ_i and μ_b are the mean of X_i and X_b, respectively.

$H_0 \equiv \mu_i = \mu_b$ is the null hypothesis, stating that S_2 is independent of S_1, and, as a result, injecting delays in S_1 does not change the request arrival distribution in S_2.

Similarly, $H_1 \equiv \mu_i \neq \mu_b$.

We compute the z-score of the 2-sample z-test using the following formula: $z = \frac{\overline{X_b} - \overline{X_i}}{\sqrt{\frac{\sigma_b^2}{n_b} + \frac{\sigma_i^2}{n_i}}}$

We already showed that if service S_2 depends on S_1, $\mu_i = \mu_0 \times (1-\rho)$ and $\mu_b = \mu_0 \times (1+\rho)$. It can be shown that, regardless of matter how small ρ is, arbitrarily small $p-value$ can be obtained given large enough samples. On the other hand, if the two services are independent, regardless the size of the sample, small $p - value$ will not be obtained.

C. Paired Wilcoxon Test

In 2-sample z-test, we did not take advantage of the fact that t_i and t_b samples are pairwise related. Another alternative is to use Wilcoxon test to test whether the paired samples of t_i and t_d are drawn from the same population. In this approach, we match t_i's to their consecutive t_b. In null hypothesis, we consider t_i's and t_b's as samples of the same population. In other words, if service S_2 does not depend on S_1, delaying requests to service S_1 should not create any changes in the distribution of the requests to S_2.

To prove that service S_2 depends on service S_1, it is sufficient to show that the number of requests received on service S_2 at t_i's does not follow the same distribution as the number of requests received at t_b's. Because the t_i's and t_b's are paired and related, we use Wilcoxon signed-rank test to calculate the z-score for the null hypothesis (that t_i's and t_b's belong to the same distribution).

We report service S_2 to depend on service S_1 if any of the two statistical tests can reject the null hypothesis.

IV. Implementation

We implemented and tested a demo of the project in a small lab. We induce the minimum amount of delay that is required to detect the perturbation. This delay time should be greater than the clock discrepancy between the delay injector and flow collector devices. The clock discrepancy in our network is less than or equal to 40 milliseconds (the computer clocks are synchronized by NTP, so we used 100 milliseconds as our delay time). We are aware of the fact, there may exist services in which 100 milliseconds of delay could cause a failure, but, these services are usually not implemented in typical TCP/IP networks and they should have their own dedicated networks, as small amount of delays/jitters are expected in regular networks.

In our demo implementation, we were able to show that the busy and idle time windows (t_b and t_i's) are easily detectable.

The delayer is in the process of application in a medium-size network.

V. Conclusions

In this paper, we presented a new approach to detect dependencies among services using traffic watermarking.

References

[1] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *In NSDI*, 2007.

[2] A. Natarajan, P. Ning, Y. Liu, S. Jajodia, and S. E. Hutchinson. NSDMiner: Automated Discovery of Network Service Dependencies. In *In proceedings of IEEE International Conference on Computer Communications (INFOCOM '12)*, 2012.

http://gswc.cs.ucsb.edu