From Patching Delays to Infection Symptoms: Using Risk Profiles for an Early Discovery of Vulnerabilities Exploited in the Wild

Abstract

At any given time there exist a large number of software vulnerabilities in our computing systems, but only a fraction of them are ultimately exploited in the wild. Advanced knowledge of which vulnerabilities are being or likely to be exploited would allow system administrators to prioritize patch deployments, enterprises to assess their security risk more precisely, and security companies to develop intrusion protection for those vulnerabilities. In this paper, we present a novel method based on the notion of community detection for early discovery of vulnerability exploits. Specifically, on one hand, we use *symptomatic* botnet data (in the form of a set of spam blacklists) to discover a community structure which reveals how similar Internet entities behave in terms of their malicious activities. On the other hand, we analyze the *risk behavior* of end-hosts through a set of patch deployment measurements that allow us to assess their risk to different vulnerabilities. The latter is then compared to the former to quantify whether the underlying risks are consistent with the observed global symptomatic community structure, which then allows us to statistically determine whether a given vulnerability is being actively exploited in the wild. Our results show that by observing up to 10 days' worth of data, we can successfully detect vulnerability exploitation with a true positive rate of 90% and a false positive rate of 10%. Our detection is shown to be much earlier than the standard discovery time records for most vulnerabilities. Experiments also demonstrate that our community based detection algorithm is robust against strategic adversaries.

1 Introduction

Most software contains bugs, and an increased focus on improving software security has contributed to a growing number of vulnerabilities that are discovered each year [10]. Vulnerability disclosures are followed by fixes, either in the form of patches or new version releases. However, the installation/deployment of software patches on millions of vulnerable hosts worldwide are in a race with the development of vulnerability exploits. Owing to the sheer volume of vulnerability disclosures, it is hard for system administrators to keep up with this process. The severity of problem was highlighted in 2017 by the the WannaCry and NotPetya outbreaks, as well as the Equifax data breach exposing sensitive data of more than 143 million consumers; in all three cases the underlying vulnerability had been patched (but not deployed) months before the incident [40, 41, 17]. Prior research suggests that, on median, at most 14% of the vulnerable hosts are patched when exploits are released publicly [25].

On the other hand, many vulnerabilities are never exploited. For instance, Nayak et al. [27] found that only 15% of known vulnerabilities are exploited in the wild. In an ideal world, all vulnerabilities should be patched as soon as they are identified regardless of their possibility of eventual exploitation. However, in reality, we live in a resource-constrained world where risk management and patch prioritization become important decisions. Even though patches may be released before or shortly after the public disclosure of a software vulnerability, many enterprises do not patch their systems in a timely manner, sometimes caused by the need or desire to test patches before deploying them on their respective machines [4]. Within this context, the ability to detect critical vulnerabilities prior to incidents would be highly desirable, as it enables enterprises to prioritize patch testing and deployment. Furthermore, identifying actively exploited-in-the-wild vulnerabilities that have not yet been addressed by the software vendor can also guide them in prioritizing patch development.

However, determining critical software vulnerabilities is non-trivial. For example, intrinsic attributes of vulnerabilities, such as the CVSS score [23], are not strong predictors of eventual exploitation [31], underlining the

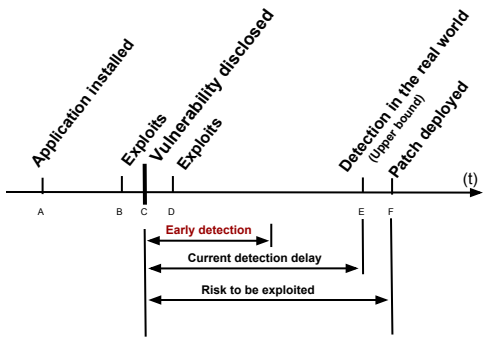


Figure 1: Vulnerability disclosure, exploits and detection time line. Early detection in this study refers to the ability to detect after t_C, i.e., post-disclosure, but much earlier than t_E, the current state of the art.

need for detection techniques based on field measurements. In this paper, we ask the question: *How early can we determine that a specific vulnerability is actively being exploited in the wild?* To put this on the appropriate time scale, we illustrate the sequence of events associated with a vulnerability in Figure 1: its introduction at t_A with application installation, disclosure at t_C, patching at t_F; and for those eventually exploited in the wild, detection at t_E. However, real exploitations may occur much earlier, shown at t_D (post-disclosure) and sometimes t_B (pre-disclosure). In this study, by early detection we refer to the ability to detect exploits after t_C (post-disclosure) but before t_E (before the current state of the art).

We show that this early detection can be well accomplished by using two datasets: end-host software patching behavior and a set of Reputation Blacklists (RBLs) capturing IP level malicious (spam) activities. Specifically, when viewed at an aggregate level, e.g., an ISP, the patching delays for a given vulnerability constitute a *risk profile* for that ISP. If a *symptom* that often follows exploitation (e.g., increased spam or malicious download activities) occurs in a group of ISPs that share a certain risk profile, then it is likely that the vulnerabilities associated with that risk profile are being exploited in the wild. We show that there is strong empirical evidence of a strong correlation between risk profiles and infection symptoms, which enables early detection of exploited vulnerabilities, even in cases where the exploit was not yet discovered and where causal connection between exploitation and the symptom is not known.

By observing these signals up to 10 days after the disclosure (t_C), we can detect exploits with true and false positive rates of 90% and 10%, respectively. Note that compared to other techniques such as detection of exploits from social media posts (which usually appear around the time exploits are discovered) [31], we base our detection on statistical evidence of exploitation from real-world measurements, which can capture much weaker indications of exploits shortly after the public disclosure of a vulnerability.

Our main contributions are summarized as follows:

1. We use a community detection [43] method for correlating and extracting features from user patching data and IP level malicious activities. We show that the resulting features can detect active exploitation, validated using a ground-truth set of vulnerabilities known to be exploited in the wild.

2. Using these features, combined with other intrinsic features of a given vulnerability, we show that accurate detection can be achieved within a few days of vulnerability disclosure. This is much earlier than the state-of-the-art on average, and thus provides significant time advantage in patch development and deployment.

3. The community structure generated during feature extraction can also be used to identify groups of hosts at risk to specific vulnerabilities currently being exploited, adding to our ability to strengthen preventative and protective measures.

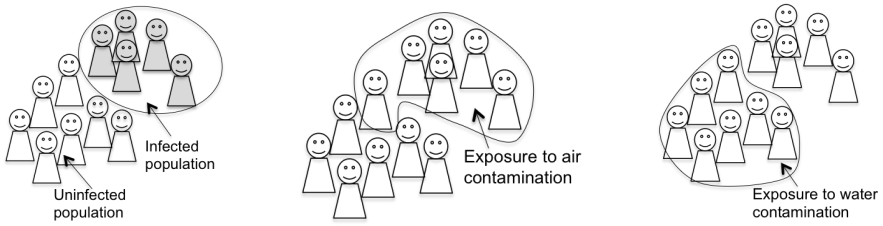
4. We evaluate the robustness of our technique against strategic adversaries, observing graciously degrading performance even when the adversary can control a significant number of hosts within many ISPs.

The rest of paper is organized as follows. In Section 2 we outline the conceptual idea behind our methodology and how community detection is used as a feature extraction tool. We describe our datasets and data processing in Sections 3 and 4. Section 5 details the community detection technique. Section 6 presents our classifier design, detection performance, and comparison with a number of alternatives. Section 7 discusses how our results could be used in practice. Section 8 summarizes related work and Section 9 concludes the paper.

2 Overview of concept and methodology

Our study is premised on a simple observation, that vulnerability exploitation leads to host infection, which then leads to manifestation of symptoms such as malicious activities. However, using the latter to detect the former is far from trivial: observed signs of infection do not reveal which vulnerability is the underlying culprit.

This led us to consider a more verifiable hypothesis: Entities (to be precisely defined shortly) that exhibit similar patching behavior in a particular vulnerability (and thus their vulnerability state) might also exhibit similar patterns of infection associated with that vulnerability



(a) Symptom pattern (b) Risk behavior 1 (c) Risk behavior 2

Figure 2: Detecting active viral strain by comparing population symptom pattern and risk behavior pattern. There are two strains of viruses: those exposed to air contamination are more at risk/susceptible to strain 1, while those exposed to water contamination are more at risk/susceptible to strain 2. By comparing the symptom group to the risk groups we can infer which strain is likely to be the underlying cause of the infection.

if it is being actively exploited; on the other hand, the same similarity association should not exist if the vulnerability is not being actively exploited. If this hypothesis holds, then it follows that one should be able to assess the strengths of association between patching behavior and infection symptoms and use it to detect whether a vulnerability is likely being actively exploited.

2.1 Main idea

We illustrate the above idea using an analogy shown in Figure 2. Suppose in any given year multiple strains of a virus may be active in a particular region. Each strain works through a different susceptibility: some through contaminated air, some through contaminated water, shown in Figure 2b and 2c respectively. When infected, regardless of the active strain, the outward symptoms are indistinguishable. However, if we know the infected population, then by identifying the underlying risk pattern it becomes possible to infer which strain may be active. Comparing Figure 2b to 2a and then 2c to 2a, we see a large overlap between the symptom group and the group at risk to strain 1 (through air contamination), indicating a likelihood that stain 1 is active; by contrast, the symptom group and those at risk to strain 2 (through water contamination) are largely disjoint, suggesting that strain 2 is likely not active.

To apply this analogy in our context, the symptom pattern refers to malicious activities while risk behavior refers to host patching. More specifically, infected population maps to hosts showing explicit signs of botnet activities, and exposure to active (non-active) viral strains maps to having vulnerabilities that are (not) being actively exploited.

2.2 Challenges

This example illustrates the conceptual idea behind our methodology, though it is a gross simplification as we elaborate below. In particular, we face two challenges. First, the telemetry that many security vendors collect on end-hosts is often anonymized, for user privacy reasons, and omits attributes that may identify the host, such as its IP address. This makes it impossible to correlate the risky behaviors (reflected in this telemetry) with symptoms (reflected in RBLs) at the host or IP level. Yet since we are correlating behavioral and symptomatic data, it is essential that both are associated with the same entity. To resolve this, we use aggregation to assess this idea at a higher level. Specifically, while the patching data does not contain IP addresses, it shows ISP information associated with each host. This allows us to aggregate patching behavior at the ISP level. On the RBL side, we use a separate IP intelligence database to aggregate malicious behavior at the ISP level by using IP to ISP mappings. In other words, each unit in the population shown in the above example now maps to an ISP. With this aggregation, the above hypothesis essentially states that ISPs behaving similarly in patching a certain vulnerability (risk patterns) are most likely to show similar infection symptoms if that vulnerability is being exploited.

This technique can be adapted for a more fine-grained aggregation, such as autonomous systems (ASs), or not using any aggregation when both risk behavior and symptoms are available at the host level. However, as is evident from our results in Section 6.2, aggregation at the ISP level is not too coarse so as to impede our technique from detecting actively exploited vulnerabilities.

Our second challenge is in determining the right metric to use to capture "similarity" both in the patching behavior and in the symptoms. Unlike what's shown in the example, in our context neither the symptoms (spam activities from an ISP) nor the risk behaviors (patching records of end-hosts in an ISP) are binary, or even necessarily countable as they are extracted from time series data. This makes identifying either pattern within the population much less straightforward. One natural first step is to compute pairwise correlation for each pair of

ISPs' time series. This results in two similarity matrices, one from the patching behavior data for a specific vulnerability, one from the symptomatic infection data (collected following that vulnerability's disclosure from spam lists). It is the second-order similarity comparison between these two matrices that is hypothesized to be able to tell apart exploited vulnerabilities from non-exploited ones. To this end, we present the use of community detection [8, 43] over the symptom similarity matrix to identify groups of similar ISPs; this is then followed by quantifying the consistency between the risk behavior similarity matrix and the detected community structure. Our results show that indeed for vulnerabilities with known exploits, this match is much stronger than that for those without known exploits.

We then use the consistency measures as features, along with a number of other intrinsic features, to train a classifier aimed at detecting exploitations.

2.3 Threat model

One type of adversaries implicit in this work are those actively exploiting software vulnerabilities. One basic assumption we adopt is that such exploitation can occur as soon as the vulnerabilities are introduced (with new version releases, etc.), though our detection framework is triggered by the official vulnerability disclosure, as indicated in Figure 1. We assume such an adversary can potentially develop and actively pursue exploits for any existing vulnerability.

A second type of adversaries we consider are those who not only seek exploitation but also have the ability to control a significant number of end-hosts so as to manipulate the patching signals we use in our detection framework. In other words, this is a type of attack (or evasion attempt) against our specific detection methodology which uses patching signals as one of the inputs. The manipulation is intended to interfere with the way we measure similarity between networks; in Section 7.2 we examine the robustness of our detection method against this type of attack.

3 Datasets

Table 1 summarizes the datasets used in this study. Since we need time-aligned measurements to compare behaviors between patching and malicious activity signals, only the overlapping time period, 01/2013-07/2014, is used in our analysis.

3.1 End-host patching

Our study draws from a number of data sources that collectively characterize the users' patching behavior, allowing us to assess their susceptibility to known vulnerabilities and exploits at any point in time. This set will also be referred to as the *risk/behavioral data*. These include the host patching data [12], the National Vulnerability Database (NVD) [28], and release notes from software vendors of the products examined in our study.

Patch deployment measurements This data source allows us to observe users' patching behavior to assess their susceptibility to known vulnerabilities. We use patch deployment measurements collected by Nappa et al. on end-hosts [25]. This corpus records installation of subsequent versions of different applications along with each event's timestamp, by mapping records of binary executables on user machines to their corresponding application versions. This data is derived from the WINE dataset provided by Symantec [12], and includes observations on hosts worldwide between 02/2008 and 07/2014. In addition, we extract the security flaws affecting each application version from the National Vulnerability database (NVD), where each vulnerability is denoted by its Common Vulnerabilities and Exposures Identifier (CVE-ID).

For each host and CVE-ID, we follow the methodology described in [33] to collect the periods of time where a host is susceptible to disclosed but unpatched vulnerabilities, through the presence of vulnerable application versions on their machines. This method involves finding the state of a host, i.e., the set of applications installed on the machine, for any point throughout the observation period, and extracting the set of disclosed vulnerabilities corresponding to those application versions from NVD. Note that a user might also install different product lines of the same application, e.g., Flash Player 10 and 11, at the same time. We will elaborate on this in Section 4.1.

For this study, we analyze user patching behavior over 7 applications with the best host coverage in our dataset, namely Google Chrome, Mozilla Firefox, Mozilla Thunderbird, Safari, Opera, Adobe Acrobat Reader, and Adobe Flash Player; we ignore hosts that have recorded less than 10 events for all of these applications. Restricted to the study period of 01/2013-07/2014, we observe 37,051 events over 30,310 unique hosts.

Vulnerability exploits As noted earlier, only a small fraction of disclosed vulnerabilities have known exploits; some exploits may remain undiscovered, but a large number of vulnerabilities are never exploited. We identify the set of vulnerabilities exploited in the real world from two sources. The first is the corpus of exploited vulnerabilities collected by Carl et al. [31]. These are extracted from public descriptions of Symantec's anti-virus signatures [37], and intrusion-protection signatures (IPS) [2]. Limiting the vulnerabilities included in our study to

Category	Collection period	Datasets
End-host patching (Risk behavior)	Feb 2008 - Jul 2014	NVD [28], patch deployment measurements [12], vendors' release notes
Malicious activity (Symptom)	Jan 2013 - Present	CBL [7] , SBL [34], SpamCop [36], UCEPROTECT [39], WPBL [42]
Vulnerability exploits (Cause)	Jan 2010- Present	Securitybid [35], Symantec's anti-virus signatures [37], intrusion-protection signatures [2]

Table 1: Summary of datasets. For this study, we use the intersection of all observation windows (01/2013-07/2014).

the above 7 products between 01/2013 to 07/2014, we curate a dataset containing 18 vulnerabilities. The second source of exploits is the SecurityFocus vulnerability database [35] from Symantec. We query all CVE-IDs extracted from NVD included in our study and obtain 44 exploited-in-the-wild (EIW) vulnerabilities. Combining all curated data sets we obtain 56 exploited-in-the-wild (EIW) and 300 not-exploited-in-the-wild (NEIW) vulnerabilities.

Software release notes To find whether a host is susceptible to a vulnerability and to address the issue of parallel product lines, we utilize the release date of each application version included in our study. For Thunderbird, Firefox, Chrome, Opera, Adobe Acrobat Reader and Adobe Flash Player, we crawl the release history logs from the official vendor's websites or a third party. However, there sometimes exist sub-versions that are not included in these sources. Thus, we also use release dates from Nappa et al. [25] who automatically extract software release dates by selecting the first date when the version appears in the patch deployment dataset [12].

3.2 Malicious activities

Our second main category of data consists of IP level spam activities and will refer to this as *symptomatic* data since malicious activities are ostensible signs that end-hosts have been infected, possibly through the use of an exploited vulnerability present on the host. This dataset is sourced from well-established monitoring systems such as spam traps in the form of various reputation blacklists (RBLs) [7, 34, 36, 39, 42]. In this study, we use 5 common user IP address based daily updated RBLs from January 2013 to July 2014 which overlap with the patch deployment measurements.

Note that the use of spam data is only a proxy for host infection caused by vulnerability exploits and an imperfect one at that. For instance, not all spam are caused by exploits; some spamming botnets are distributed through malicious attachments. Similarly, it is also common for cyber-criminals to rent pay-per-install services to install bots. In both cases, the resulting spam activities are not correlated with host patching patterns. This raises the question whether these other types of operations may render our approach ineffective. Our results show the opposite; the detection performance we are able to achieve suggests that spam is a very good proxy for this purpose despite the existence of non-vulnerability related spamming bot distributions.

Note that hosts in our patch deployment dataset are anonymized, but can be aggregated at the Internet Service Provider (ISP) level. Hence, we also use the Maxmind GeoIP2ISP service [24] (identifying 3.5 million IPv4 address blocks belonging to 68,605 ISPs) to aggregate malicious activity indicators at the ISP level. We then align the resulting time series data with aggregated patching signals for evaluating our methodology.

4 Data Processing and Preliminaries

In this section we further elaborate on how time series data are aggregated at the ISP level and how we define similarity measures between ISPs.

4.1 Aggregating at the ISP level

The mapping from hosts to ISPs is not unique; as devices move it may be associated with different IP addresses and possibly different ISPs. This is the case with both the patching data and the RBLs and our aggregation takes this into account by similarly mapping the same host to multiple ISPs whenever this is indicated in the data.

Aggregating the RBL signals at the ISP level is relatively straightforward. Each RBL provides a daily list of malicious IP addresses, from which we count the total number of unique IPs belonging to any ISP. Formally, let $R_n(t)$ denote the total number of unique IPs listed on these RBLs on day t that belong to ISP n (by mapping the IPs to prefixes associated with this ISP). This is then normalized by the size of ISP n; this normalization step is essential as pairwise comparisons between ISPs can be severely skewed when there is a large difference in their respective sizes. The normalized time series $r_n(t)$ will also be referred to as the *symptom signal* of ISP n.

Aggregating the patching data at an ISP level is significantly more involved. This is because the measurements are in the form of a sequence of application versions installed on a host with their corresponding timestamps. To

quantify the risk of a given host, we first extract known vulnerabilities affecting each application version from NVD using the Common Vulnerabilities and Exposures Identifier (CVE-ID) of the vulnerability. Each vulnerability will also be referred to as a CVE throughout this paper. However, this extraction is complicated by the fact that there may be multiple product lines present on a host, or when a user downgrades to an earlier release. Moreover, multiple product lines of a software are sometimes developed in parallel by a vendor, all of which could be affected by the same CVE, e.g,. Flash Player 10 and 11. It follows that if a host has both versions, then updating one but not the other will still leave the host vulnerable. In this study, we use the release notes described in Section 3.1 as an additional data source to distinguish between parallel product lines, by assuming that application versions belonging to the same line follow a chronological order of release dates, while multiple parallel lines can be developed in parallel by the vendor. This heuristic allows us to discern different product lines of each application and users that have installed multiple product lines on their respective machines at any point in time, leading to a more accurate estimate of their states.

We quantify the vulnerability of a single host h to CVE j on day t by counting how many versions present on the host on day t are subject to this CVE. Denoted by $W_h^j(t)$, in most cases, this is a binary indicator (either there is a single version subject to this CVE or there isn't), but occasionally this can be an integer > 1 due to the presence of parallel product lines mentioned above. This quantity is then summed over all hosts belonging to an ISP n, resulting in a total count of unpatched vulnerabilities present in this ISP. We again normalize this quantity by the ISP's size and denote the normalized signal by $w_n^j(t)$.

We have now obtained two types of time series for each ISP n: $r_n(t)$ denoting the normalized malicious activities (also referred to as the symptom signal), and $w_n^j(t)$, $j \in \mathscr{V}$, denoting the normalized risk with respect to CVE j; the latter is a set of time series, one for each CVE in the set $\mathscr{V}$ (also referred to as the risk signal). Note that $r_n(t)$ is not CVE-specific; however, a given CVE determines the time period in which this signal is examined as we show next.

4.2 Similarity in symptom and in risk

As described in the introduction and highlighted in Figure 2, our basic methodology relies on identifying the similarity structure using symptom data and quantifying how strongly the risk patterns are associated with the symptom similarity structure. This is done for each CVE separately. Note that our aggregated malicious activity signal $r_n(t)$ for ISP n is agnostic to the choice of CVE, since the observed malicious activities from a single host can be attributed to a variety of reasons including various CVEs the host is vulnerable to. However, our analysis on a given CVE determines the time window from which we examine this signal. Specifically, consider the following definition of correlation between two vectors $u[0:d]$ and $v[0:d]$, which tries to find similarity between the two vectors by allowing time shifts/delays between the two:

$$S_{u,v}(k) = \frac{\sum_{t=k}^{d} u(t) \cdot v(t-k)}{\sqrt{\sum_{t=0}^{d-k} v(t) \cdot v(t) \cdot \sum_{t=0}^{d-k} u(t+k) \cdot u(t+k)}}, \quad (1)$$

where $k = 0, \cdots, d$ denotes all possible time shifts. The above equation keeps v fixed and slides u one element at a time and generates a sequence of correlations over increasingly shorter vectors. Similarly, we can keep u fixed and slide v one element at a time, which gives us $S_{v,u}(k)$ for $k = 0, \cdots, d$. Our pairwise similarity measure is defined by the maximum of these correlations subject to a lower bound on how long the vector should be:

$$S_{u,v} = \max\Big(\max_{0 \le k \le d-a}(S_{u,v}(k)), \max_{0 \le k \le d-a}(S_{v,u}(k)), \quad (2)$$

where a is a lower bound to guarantee the correlation is computed over vectors of length at least $d-a$ to prevent artificially high values. In our numerical experiment a is set to $\lceil \frac{d}{4} \rceil$.

With the above definition, the pairwise symptom similarity between a pair of ISPs n and m for CVE j can now be formally stated. Assume t_o^j to be the day of disclosure for CVE j. We will focus on the time period from disclosure to d days after that, as we aim to see whether by examining this period we can detect the presence of an exploit[1]. For simplicity of presentation, we shift t_o^j to origin, which gives us two symptom signals of length $d+1$: $r_n[0:d]$ and $r_m[0:d]$, and a pairwise symptom similarity measure $S^j_{r_n,r_m}$ using Equations (1) and (2).

We can similarly define the pairwise risk similarity between this pair of ISPs, given by $S^j_{w_n,w_m}$.

5 Comparing Symptom Similarity to Risk Similarity

In this section, we first use community detection methods [8, 43] to identify the underlying communities in the pairwise symptom similarities. We then detail our technique for quantifying the strength of association between symptoms and risk behavior for specific CVEs.

[1] In Section 6 we also examine whether signs of infection can be detected *before* the official disclosure; in that case this window starts d_1 days before the disclosure and ends d_2 days after.

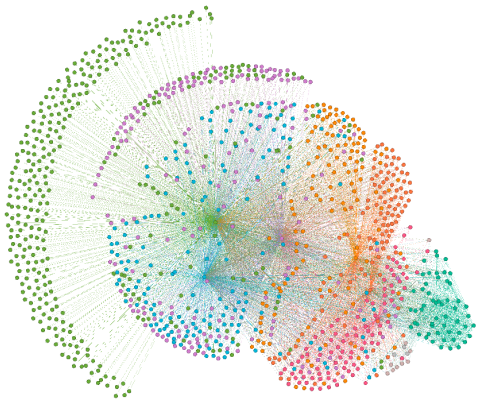
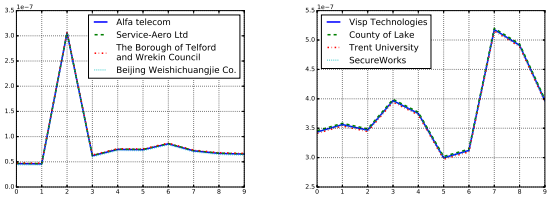


Figure 3: Visualization of community structure of malicious ISPs; each color denotes a single community.



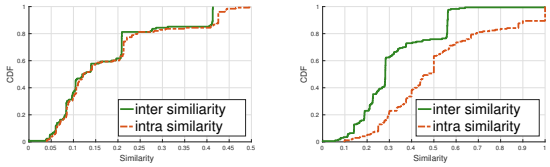
(a) The green community. (b) The pink community.

Figure 4: Aggregate malicious signals of selected ISPs belonging to either green or pink community in Fig. 3.

5.1 Community detection over symptom similarity

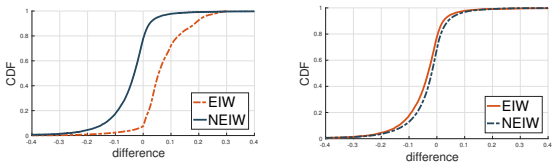
The set of pairwise similarity measures $S^j_{r_n,r_m}$ constitute a similarity matrix denoted by $\mathbf{S}^j[n,m], \forall n,m \in \mathscr{I}$ where $\mathscr{I}$ denotes the set of all ISPs included in the following analysis. This matrix is equivalently represented as a weighted (and undirected) graph, where $\mathscr{I}$ is the set of vertices (each vertex being an ISP) and the pairwise similarity $S^j_{r_n,r_m}$ is the edge weight between vertices n and m (note each edge weight is a number between 0 and 1). A community detection algorithm can then be run over this graph to identify hidden structures.

The general goal of community detection is to uncover hidden structures in a graph; a typical example is the identification of clusters (e.g., social groups) that are strongly connected (in terms of degree), whereby nodes within the same cluster have a much higher number of in-cluster edges than edges connecting to nodes outside the cluster. This has been an extensive area of research within the signal processing and machine learning community and has found diverse applications including biological systems [15, 38, 44], social networks [20, 43, 44], influence and citations [26, 43, 44], among others.



(a) CVE-2014-1504 (NEIW). (b) CVE-2014-0496 (EIW).

Figure 5: Intra- and inter-cluster risk similarity on different types of CVEs based on community detection.



(a) Difference between intra- and inter-cluster risk similarity over detected communities.

(b) Difference between intra- and inter-cluster risk similarity over a random partition of ISPs.

Figure 6: Distinguishing between EIW and NEIW CVEs.

In our context, the similarity matrix $\mathbf{S}^j[n,m]$ induces a weighted and fully-connected graph. The result of community detection over such a graph is a collection of clusters, each of which represents ISPs that share very similar symptoms. We use two state-of-the-art community detection algorithms, both of which detect overlapping communities, i.e., a node may belong to multiple clusters. The first one is BigClam (Cluster Affiliation Model for Big Networks) [44]; this is a model-based community detection algorithm that finds densely overlapping, hierarchically nested, as well as non-overlapping communities. The second is DEMON (Democratic Estimate of the Modular Organization of a Network) [8], which discovers communities by using local properties.

Figure 3 visualizes the communities discovered from the symptom similarity matrix corresponding to CVE-2013-2729 from 2013/05/16 to 2013/05/26 using the Force Atlas layout [21] provided by [3]; different colors encode different communities identified by the algorithm. In this example, an original graph of 8,742 nodes was reduced to one with 1,112 nodes and 10 detected communities.[2] To convey a sense of what the notion of community captures, we further plot the spam signals of groups of ISPs each belonging to one of two communities in Figure 4; as can be seen, those in the same community exhibit similar temporal signals.

[2]The reduction in number of nodes is due to deletion of all edges to some nodes when all their edge weights are below a certain threshold.

5.2 Measuring the strength of association between risk and symptoms

We now verify the hypothesis stated in the introduction; that is, if a CVE is being actively exploited, then ISPs showing similar vulnerabilities to this CVE are also likely to exhibit similar infection symptoms, while on the other hand if a CVE is not actively exploited, then the similarity in vulnerabilities may not be associated with similarity in symptoms. Toward this end, we note that there isn't a unique way to measure the strength of association in these two types of similarities. One could, for instance, try to directly compare the two similarity matrices $\mathbf{S}^j[r_n, r_m]$ and $\mathbf{S}^j[w_n, w_m]$; we shall use one version of this whereby we perform row-by-row correlation between the two matrices as one of the benchmark comparisons presented in Section 6.

Below we will consider a more intuitive measure. We first use the communities detected by symptom similarity to sort pairwise risk similarity values into two distinct groups: inter-cluster similarity and intra-cluster similarity. Specifically, denote the set of clusters identified by community detection over matrix $\mathbf{S}^j[r_n, r_m]$ as $\mathscr{C}$. Then if we can find a cluster $C \in \mathscr{C}$ such that both $n, m \in C$, then $S^j_{w_n,w_m}$ is sorted into the intra-cluster group; otherwise it is sorted into the inter-cluster group. This is repeated for all pairs $n, m \in \mathscr{I}$. Figure 5 shows the distribution of these values within each group for two distinct CVEs, one is known to have an exploit in the wild (detected by Symantec 20 days post-disclosure), and the other has no known exploits in the wild.

The difference between the two is both evident and revealing: for the CVE without a known exploit, Figure 5a shows virtually no difference between the two distributions, indicating that the risk similarity values are not differentiated by the symptom patterns. On the other hand, for the exploited CVE (though only known after the analyzed observation time period), Figure 5b shows a very distinct difference (p value of Kolmogorov-Smirnov test < 0.01) between the two groups. In particular, the intra-cluster group contains much higher risk similarity values. This suggests that high risk similarity coincides with high symptom similarity (which is what determined the community structure). Also worthy of note is the fact that for the exploited CVE, the earliest date of exploit observation on record is 20 days post-disclosure (disclosure on 01/15/2014, observation in the wild on 02/05/2014), whereas this analysis is feasible within 10 days of the disclosure (01/15-01/25/2014). This suggests that exploits occur much sooner than commonly reported, and that early detection is possible.

We sum up the values in each group and take the difference between the intra-cluster sum and the inter-cluster sum and denote it by D^j. This allows us to evaluate the entire set of CVEs. Figure 6a shows the distribution of D^j over two CVE subsets: one with known exploits (with observation dates at least 10 days post-disclosure) and one without known exploits. We see that for the group of exploited CVEs, the intra-cluster risk similarities are decidedly higher, suggesting a consistency with communities detected using symptoms. By contrast, for non-exploited CVEs, there is no appreciable difference between the two groups; indeed the distribution looks very similar to that obtained using random partitions of the ISP shown as a reference in Figure 6b. We also plot for each CVE its time to earliest detection on record against the above similarity difference measure in Figure 7; the curve highlights the mean of D_j in each delay bin. We observe a general downward trend in the mean, i.e., for exploits spotted earlier their inter-cluster and intra-cluster similarity difference is also more pronounced. This is consistent with our belief that the similarity difference is fundamentally a sign of active exploitation, which coincides with being detected earlier; for those detected much later on, it is more likely that some of them occurred later and therefore could not be observed during the early days.

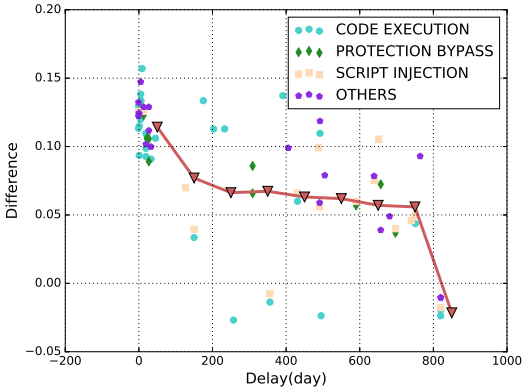


Figure 7: Time to recorded detection (x-axis) vs the difference measure calculated within 10 days post disclosure (y-axis); the red curve is the mean difference within each delay bin. Different categories of CVEs are color-coded, with ties broken randomly when a CVE belong to multiple categories.

6 Early detection of exploits in the wild

Our results in the previous section shows that the intra- and inter-cluster risk similarity distributions as well as the difference D^j are statistically meaningful in separating one group of CVEs (exploited) from another (not exploited). This suggests that these can be used as features in building a classifier aiming at exploits detection.

6.1 Features and labels

Each CVE in our sample set is labeled as either exploited or un-exploited, which constitutes the label. As describe in Section 3.1, our ground-truth comes from three sources, public descriptions of Symantec's anti-virus signature, intrusion-protection signatures and exploit data from SecurityFocus. Each CVE also comes with a set of features. In addition to the spam/symptom data and patching/risk data we analyzed rigorously in the previous section, we will also use intrinsic attributes associated with each CVE extracted from NVD.

Specifically, CVE summary information offers basic descriptions about its category, the process to exploit it, and whether it requires remote access, etc. These are important static features for characterizing the properties of a CVE. We apply bag of words to retrieve features from the summaries after punctuation and stemming processes. In total we obtained 3037 keywords from our data set. We then select 16 features with the highest mutual information with labels; these are shown in Table 2. We observe that keywords such as `attack`, `exploit`, `server`, and `allow`, have higher mutual information with labels of exploited, which is consistent with common understanding of what might motivate exploits.

Below we summarize the complete set of features used in this study (each family is given a category name), some of which are introduced for comparison purposes as we describe in detail next.

- **[Community]:** The difference in distribution (intra-cluster minus inter-cluster similarity) shown in Figure 5, in the form of histograms with 20 bins.
- **[Direct]:** The distribution of row-by-row correlation between the two similarity matrices $\mathbf{S}^j[r_n, r_m]$ and $\mathbf{S}^j[w_n, w_m]$, in the form of 20-bin histograms.
- **[Raw]:** The two similarity matrices $S^j[r_n, r_m]$ and $S^j[w_n, r_m]$.
- **[Intrinsic]:** The top 20 intrinsic features using bag of words as shown in Table 2.
- **[CVSS]** CVSS [23] metrics and scores. For each CVE, we use three metrics: AcessVecotr, AccesComplexity, and Authentication, which measure the exploit range, required attack complexity and the level of authentication needed for successful exploitation, respectively

We can also categorize these sets of features as graph-based ([Community], [Direct], [Raw]) and intrinsic ([Intrinsic], [CVSS]) features. The intrinsic features describe what is known about a vulnerability at the time of disclosure, e.g., whether it can be used to gain remote control of the host. Intuitively, these features can affect the likelihood of a CVE being targeted by cyber-criminals. On the other hand, graph-based features can detect the onset of active exploitation, by associating similar patching behavior with similarity in infection patterns. Our results in the following section demonstrate that while intrinsic features alone are poor predictors of eventual exploitation of a CVE, combining intrinsic attributes with graph-based features enables early and accurate detection of EIW vulnerabilities.

Keyword	MI Wild	Keyword	MI Wild
affect	0.0006	allow	0.0045
attack	0.0069	crafted	0.0012
corruption	0.0019	google	0.0012
dll	0.0016	free	0.0016
function	0.0012	exploit	0.0016
server	0.0020	runtime	0.0047
remote	0.0004	memory	0.0001
service	0.0008	xp	0.0004

Table 2: The top 16 intrinsic features, and their mutual information with both sources of ground-truth data.

6.2 Detection performance

We now compare the detection performance by training classifiers using different subsets of the features listed above. In training the classifiers, we note there is an imbalance between our EIW (56) and NEIW (300) classes of CVEs. For this reason, the training and testing are conducted using 20 rounds of random sub-sampling from the NEIW set to match its size with the EIW set; for each round, we apply 5-fold cross validation to split the dataset into training and test sets. We train Random Forests [29] for classification, and average our results over all 20 rounds; our results are reported below.

When using [Community] features, we observed similar performance for BigClam and DEMON. BigClam has linear time complexity, so for simplicity of exposition, below we only report our results using BigClam.

Also for comparison, we will directly use the two similarity matrices (the [Raw] features) to train a classifier. The dimensionality of the matrix is equal to the number of valid ISPs, 3050 by 3050. This is much higher in dimension than the number of instances we have, leading to severe overfitting if used directly. We thus apply a common univariate feature selection method [32] provide by [29] to obtain $K = 150$ features with the highest values based on the chi-squared test[3]. Three standard machine learning methods are then used to train a

[3]We did not use PCA for dimensionality reduction in order to retain interpretability of the features used.

classifier: SVM, Random Forest and a fully-connected neural network with three hidden layers and 30 neurons for each hidden layer. We observe similar performance for all examined models, and thus we only report our results using Random Forest classifiers. We depict the ROC (Receiver Operating Characteristic) curves and report the AUC (Area Under the Curve) score as performance measures. We train and compare multiple classifiers on different sets of features:

- "All features": This is a classifier trained with all features using 10 days of data post-disclosure.
- "Community features": A set of classifiers trained using only [Community] and on 10 days of observational data post-disclosure (for both symptom and risk). Only CVEs whose known detection dates are beyond 10 days are used for testing these classifiers.
- "Direct features": Trained based on the [Direct] features alone on 10 days of data post-disclosure.
- "Raw features": Trained with [Raw] features on 10 days of observational data post-disclosure.
- "Day x": A set of classifiers trained using only [Community] and on x days of observational data post-disclosure (for both symptom and risk). Only CVEs whose known detection dates are beyond Day x are used for testing these classifiers.
- "Back x": A set of classifiers trained using only [Community] and on 10 days of observational data starting from x days *before* disclosure (for both symptom and risk).
- "Intrinsic features": Trained using [Intrinsic] and [CVSS] families of features.
- "Community+Intrinsic": This is a classifier trained with [Intrinsic] and [Community] features on 10 days of observational data post-disclosure.
- "Direct+Raw+Intrinsic": This is a classifier trained with [Intrinsic], [Direct], and [Raw] features on 10 days of observational data post-disclosure.

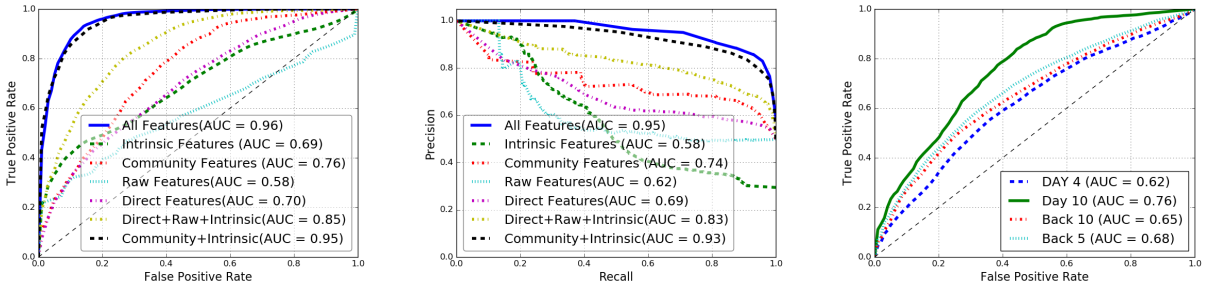
The main comparison is given in Figure 8a. We see a remarkable improvement in detection performance when we combine the community features with CVE intrinsic features. We see that even though both community and direct features are extracted from the raw features, they both perform much better than directly using raw features. In particular, the community detection based method is shown to perform the best among these three. The reason why extracted features perform better than raw features is because with the latter a lot of the temporal information embedded in the time series data is underutilized (e.g., in decision tree type of classifiers, time series data are taken as multiple individual inputs), whereas the features we extract (either in the form of community comparison or in the form of row-by-row correlation) attempt to preserve this temporal information.

Additionally, we see that combining community features with intrinsic features achieves very good detection performance, almost similar to the concatenation of all features; this suggests that when combined with intrinsic features, community features can effectively replace the use of raw and direct features. Finally, the overall attainable performance is very promising: 96% AUC, and 90% and 10% true and false positive rates. The same set of results are re-plotted in terms of precision and recall in Figure 8b.

As mentioned earlier and observed here, the intrinsic features by themselves are not particularly strong predictors, and weaker than the community features when used alone, as measured by AUC. This is because the intrinsic features are *a priori* characterizations of a vulnerability (thus the use of which amounts to prediction), whereas community features are *posteriori* signs of exploitation, allowing us to perform detection. It is thus not surprising that the latter is a more powerful metric. It is however promising to see that the two sets of features complement each other well, resulting in much higher performance when combined.

It should be noted that this level of performance still falls short of what could be attained in a typical intrusion detection systems (IDS) or spam filters, and there are a few reasons for this. Firstly, as mentioned earlier our labeling of vulnerabilities as exploited and non-exploited may be noisy: some exploited vulnerabilities may have remained unidentified and unreported. Secondly, in an IDS type of detection system there are typically very specific signatures one looks for, whereas in our setting the analysis is done over large populations where such signatures become very weak or non-existent; e.g., we can only observe if a host is sending out spam without any visibility into how or why. Accordingly, a performance gap is expected if comparing to IDS type of detection systems. It is however worth noting that in our setting a false positive is not nearly as costly as one in an IDS; ours would merely suggest that an as-yet unexploited CVE should be prioritized for patch development/deployment, which arguably would have to be done at some point regardless of our detection result.

If multiple CVEs are simultaneously exploited, our detection can still work as long as the hosts have non-identical patching behavior for these CVEs. This is because the risk behavior would be different even if the infection groups are the same, as we showed in Figure 2c.



(a) (b) (c)

Figure 8: ROC and AUC comparison (left), precision and recall comparison (center), and Comparison between different observation windows post- and pre-disclosure (right).

If the host population also exhibit the same patching behavior toward these CVEs, then the resulting ambiguity will cause our algorithm to "detect" all of these CVEs, only one/some of which are the culprit. This would be another type of false positive; the consequence however is again limited – all these CVEs will be suggested as high priority even though one or some of them could have waited.

Note that the accuracies presented here are obtained in spite of multiple sources of noise that can appear in our datasets or imperfections in our methodology. For instance the one-to-multiple mapping from symptoms of malicious behavior (indicated by RBLs) to vulnerabilities, especially when multiple vulnerabilities appear in the same time window, and hosts appearing in a blacklist for reasons other than exploitation of software vulnerabilities, can introduce noise in the measured symptoms (malicious activities). Furthermore, aggregation at a coarse level can lead to only observing the averages of behavior that could otherwise be utilized to detect exploitation. However, the ground-truth for testing the performance of our technique is independent of the aforementioned sources of noise, and the observed performance shows that our method is, to a large extent, robust to these imperfections.

We next examine the impact of the length of the observational period when using community detection, by comparing the ROCs of classifiers trained using different number of days, immediately following disclosure, as well as starting from a few days before disclosure. This is shown in Figure 8c. We see that as we increase the observation period post-disclosure the predictive power of the similarity comparison improves. This is to be expected as longer periods are more likely to capture symptoms of infection especially during the early days as vulnerabilities are just starting to be exploited. Interestingly, starting the observation even before disclosure also seems to be picking up information, an indication that some exploits do start earlier than official disclosure as mentioned in the introduction. Among the examined set the "Day 4" version is the worst-performing; this is due to a very short window of observation, only 4 days post-disclosure. This short window affects the effectiveness of time series data analysis but also is more likely to miss information that is just emerging post-disclosure.

7 Case Studies and Discussion

In this section we present a few examples of our system's output for (potentially) zero-day EIW vulnerabilities, and discuss the robustness of our technique against strategic attackers, and its practical utility for building real-world monitoring of software vulnerabilities.

7.1 Case studies

Figure 8c suggests that by performing a retrospective analysis on the disclosure date, our technique can also detect zero-day exploits. We now discuss two such examples below, both of which were detected by the "Back 10" classifier with an operating point (corresponding to a threshold of 0.7) of 80.6% true positive and 20% false positive rate.

CVE-2013-0640 This vulnerability affects Adobe Acrobat Reader and was disclosed on 02/13/2013 [1]. It allows remote attackers to execute arbitrary code via a crafted PDF document. Our system detected this vulnerability on the same day as disclosure using data from the preceding 10 days. Interestingly, we also found proof of zero-day exploits for this vulnerability [5].

CVE-2013-5330 This vulnerability affects several versions of Adobe Flash Player and was disclosed on 11/12/2013. It allows attackers to execute arbitrary code

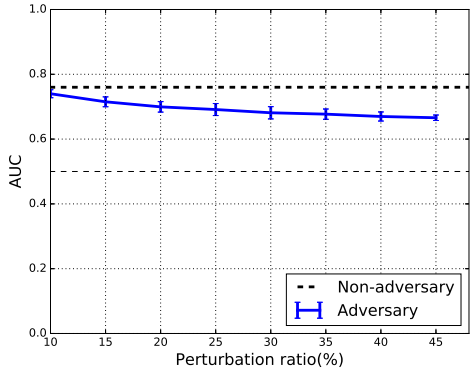


Figure 9: Robustness of performance against an adversary controlling hosts within a percentage of all ISPs (x-axis).

or cause a denial of service (memory corruption) via unspecified vectors. Again, our system detected that this vulnerability on the disclosure day using data from the preceding 10 days. While this vulnerability has been reported as exploited in the wild, the earliest report was on 01/28/2014 [9]; our results suggest that this CVE might have been exploited months before this date.

7.2 Robustness against strategic attacks

In security applications, strategic adversaries always have incentive to manipulate instances they have control over to evade detection. During such manipulations, the attacker usually needs to mimic normal user behavior as well as preserving their original malicious functionality without making arbitrarily large changes. Since our detection method relies on models trained using measurement data, it is potentially vulnerable to attempts of data manipulation. An adversary of the second type mentioned in Section 2.3 is such an example: we assume it has the ability to alter the patching information (as it is collected) from a significant number of hosts, so as to alter the aggregate signals and skew the similarity analysis. Below we examine how robust our detection system is against such evasion attempts.

We will simulate this data manipulation by altering the risk signals for a group of ISPs. Specifically, we randomly select a set of N ISPs from the total population $\mathscr{I}$ and revise their risk signals as follows:

$$w_n^j(t) \longleftarrow \frac{\sum_{i\in N} w_i^j(t)}{\|N\|} \pm \gamma \cdot w_n^j(t),\ n \in N, \qquad (3)$$

where the first term is the average value among this controlled group of ISPs, and γ is randomly drawn from the set $(0.1, 0.2, 0.3)$ for each n (similarly, $\pm$ is determined by a random coin flip) to serve as a small perturbation around the average. The intention of this manipulation is to make these N values very similar to each other, each a small perturbed version of the common average; this revision also preserves the original average so as to minimize the likelihood detection by a simple statistical test.

For each selection N we perform 20 random trials of the detection algorithm, each over different random perturbations shown above. The average AUC is reported in Figure 9 as we increase the size of N, from 10% to 45% as a fraction of the overall ISP population $\mathscr{I}$. As can be seen, our method is fairly robust against this type of evasion attacks with gracefully degrading performance. It should be noted that even at 10% this would have been an extremely costly attack as it indicates the control of hosts within hundreds of ISPs.

7.3 Practical use

We next discuss how the proposed methodology could be used in practice, in real time, and by whom. Any software or AV vendor, as well a security company would perform such a task; they typically have access to data similar in nature to WINE. The RBLs and NVD are publicly available, so is IP intelligence data (usually at a cost). Since we rely on CVE information to perform user patching data aggregation (risk with respect to specific vulnerabilities) and on intrinsic features of a vulnerability, the detection process is triggered by a CVE disclosure. Following the disclosure, malicious activity data and user patching data can be processed on a daily basis. On each day following the disclosure we have two signals of length: risk signal $w(t)$ and symptom signal $r(t)$ for each ISP. Community detection, feature extraction, and detection then follow as we described earlier. In addition, the community structure can be updated in an online fashion, so the information can be obtained and maintained in a computationally efficient manner [22].

How our detection system can enhance security in practice lies in the primary motivation of this study: [33] has measured the portion of the vulnerability window (time from disclosure to patch installation) that is incurred by user negligence for four of the products included in this paper, the largest is roughly 60% for Chrome and Flash Player; suggesting delays may exist in patch development. The ability to detect active exploits early would allow a software vendor to better prioritize patch development and more judiciously allocate its resources. A secondary use of the system is to allow a network (e.g., an ISP) to identify its most at-risk host populations that have not patched a vulnerability with detected exploits, and encourage prompt actions by these hosts. This system is not meant to alter individual end-user patching behavior, but would allow users through

silent updates to get patches sooner for vulnerabilities most at risk of being exploited.

Furthermore, [25] suggests that in the timeline of evolution of software patches, patch development happens soon after vulnerability disclosure, yet there is a gap prior to patch deployment, as, e.g., enterprises want to test patches before they deploy them. In this landscape, early detection can also be utilized by enterprises to prioritize patch deployment of vulnerabilities that are being actively exploited. Our community detection method can be used to complement intrinsic attributes of a CVE, such as the CVSS score, to detect critical vulnerabilities with more precision.

Finally, while our technique is evaluated over measurements that are 3-4 years old (due to unavailability of the WINE dataset), the updating mechanism employed by the software examined herein have remained largely the same. In particular, except for Adobe Acrobat Reader, all of the software included in this study were using silent updates to automatically deliver patches to users, at the start of our observation windows (1/2013). This supports our claim that the same dynamics apply to more recent vulnerabilities, where even though patches are developed and disseminated by vendors through automated mechanisms, users and enterprises often opt out of keeping their software up to date, leading to eventual exploitation, and then followed by observation of symptoms. WannaCry and NotPetya outbreaks (exploiting CVE-2017-0144), and the Equifax data breach (caused by CVE-2017-5638) are all recent examples of this phenomena, where patches for the underlying vulnerabilities had been disseminated by software vendors months before each incident, but had not yet been deployed on the compromised machines [40, 41, 17].

8 Related Work

Bozorgi et al. [6] used linear support vector to predict the development of proof-of-concept (POC) exploits by leveraging exploit metadata. Our interest in this study is solely on exploits in the wild and their early detection. In [31] social media was used to predict official vulnerability disclosure and it was shown that accurate prediction can be made to gain a few days in advance of disclosure announcements as an effective means of mitigating zero-day attacks. The focus of this study by contrast is the detection of exploits post-disclosure by using two distinct datasets, one capturing end-host patching behavior, the other IP level malicious activities.

Prior studies on end-host patching behavior heavily focus on understanding the patching behavior itself and its implication on user vulnerability and how it decays/evolves over time; these include e.g., observing patching patterns at different stages [30], the decay rate [13, 45], patching behavior across different update mechanisms [11, 16], vulnerability decay and threat by shared libraries [25], among others. To the best our knowledge, ours is the first study that attempts to to detect active exploitation by correlating patching behavior and vulnerability data with host infection data inferred from a set of spam blacklists.

Detection of community structures in graphs or networks is an increasingly active field in graph mining and has seen extensive work, see e.g., [14]. It has found wide applications in sociology [20, 43, 44], biology [15, 38, 44], computer science [26, 43, 44], and many other disciplines, where data is often modeled as a graph, using community detection as a tool for visualization, to reduce graph size, and find hidden patterns. As an example, the notion of similarity graphs is a commonly used technique to represent data. For instance, Holm et al. built similarity protein graphs where nodes represent protein structures and edges represent structural alignments for efficient search in the protein structure databases [19]. Similarly, to find disjoint subsets of data, E. Hartuv et al. created similarity graphs on pairs of elements, where similarity is determined by the set of features for each element, and then perform clustering on them [18]. In this study, we build similarity graphs among ISPs by measuring the similarity between their time series data.

9 Conclusion

We presented a novel method based on the notion of community detection to perform early detection of vulnerability exploitation. We used symptomatic botnet data to discover community structure, which reveals how similar network entities behave in terms of their malicious activities. We then analyzed the risk behavior of end-hosts through a set of patching data that allows us to assess their risk to different vulnerabilities. The latter was then compared to the former to quantify whether the underlying risks are consistent with the observed global symptomatic community structure, which then allowed us to statistically determine whether a given vulnerability is being actively exploited. Our results show that by observing up to 10 days' worth of data post-disclosure, we can successfully detect the presence of exploits at 90% accuracy. This is much earlier than the recorded times of detection for most vulnerabilities. This early detection capability provides significant time advantage in patch development and deployment, among other preventative and protective measures. The community structure generated during the feature extraction can also be used to identify groups of hosts at risk to specific vulnerabilities currently being exploited, adding to our ability to strengthen preventative and protective measures.

References

[1] Adobe. Security advisory for adobe reader and acrobat. https://www.adobe.com/support/security/advisories/apsa13-02.html.

[2] Symantec attack signatures. http://bit.ly/1hCw1TL.

[3] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: An open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[4] S. M. Bellovin. Patching is hard. https://www.cs.columbia.edu/~smb/blog/control.

[5] J. T. Bennett. It's a kind of magic. https://www.fireeye.com/blog/threat-research/2013/02/its-a-kind-of-magic-1.html.

[6] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *KDD*, Washington, DC, Jul 2010.

[7] CBL. http://cbl.abuseat.org/.

[8] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. DEMON: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 615–623. ACM, 2012.

[9] CVE-2013-5330 (Flash) in an unknown exploit kit fed by high rank websites. http://malware.dontneedcoffee.com/2014/02/cve-2013-5330-flash-in-unknown-exploit.html.

[10] CVE ID syntax change. https://cve.mitre.org/cve/identifiers/syntaxchange.html.

[11] T. Dübendorfer and S. Frei. Web browser security update effectiveness. In *CRITIS Workshop*, September 2009.

[12] T. Dumitraș and D. Shou. Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In *EuroSys BADGERS Workshop*, Salzburg, Austria, Apr 2011.

[13] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of Heartbleed. In *Proceedings of the Internet Measurement Conference*, Vancouver, Canada, Nov 2014.

[14] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[15] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[16] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic. Planet scale software updates. In *SIGCOMM*, pages 423–434, 2006.

[17] D. Goodin. Failure to patch two-month-old bug led to massive Equifax breach. https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug.

[18] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information processing letters*, 76(4):175–181, 2000.

[19] L. Holm, S. Kääriäinen, P. Rosenström, and A. Schenkel. Searching protein structure databases with DaliLite v. 3. *Bioinformatics*, 24(23):2780–2781, 2008.

[20] P. Hui, E. Yoneki, S. Y. Chan, and J. Crowcroft. Distributed community detection in delay tolerant networks. In *Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture*, page 7. ACM, 2007.

[21] M. Jacomy. Force-atlas graph layout algorithm. *URL: http://gephi. org/2011/forceatlas2-the-new-version-of-our-home-brew-layout*, 2009.

[22] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117, 2009.

[23] S. Lin, M. Yeh, and C. Li. Common vulnerability scoring system version 2 calculator. *PAKDD 2013 Tutorial*, 2013.

[24] MaxMind. http://www.maxmind.com/.

[25] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitraș. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *IEEE Symposium on Security and Privacy*, San Jose, CA, May 2015.

[26] N. Natarajan, P. Sen, and V. Chaoji. Community detection in content-sharing social networks. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 82–89. ACM, 2013.

[27] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitraș. Some vulnerabilities are different than others: Studying vulnerabilities and attack surfaces in the wild. In *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses*, Gothenburg, Sweeden, Sep 2014.

[28] NIST. National Vulnerability Database. `https://nvd.nist.gov`. Retrieved on 23 May 2016.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[30] T. Ramos. The laws of vulnerabilities. In *RSA Conference*, 2006.

[31] C. Sabottke, O. Suciu, and T. Dumitraș. Vulnerability disclosure in the age of social media: Exploiting Twitter for predicting real-world exploits. In *USENIX Security Symposium*, Washington, DC, Aug 2015.

[32] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[33] A. Sarabi, Z. Zhu, C. Xiao, M. Liu, and T. Dumitraş. Patch me if you can: A study on the effects of individual user behavior on the end-host vulnerability state. In *International Conference on Passive and Active Network Measurement*, pages 113–125. Springer, 2017.

[34] The Spamhaus project: SBL, XBL, PBL, ZEN lists. `http://www.spamhaus.org/`.

[35] Symantec security focus community, 30 March 2013. `http://www.securityfocus.com/`.

[36] SpamCop blocking list. `http://www.spamcop.net/`.

[37] Symantec Corporation. Symantec threat explorer. `http://www.symantec.com/security_response/threatexplorer/azlisting.jsp`, 2012.

[38] G. S. Thakur. Community detection in biological networks. *Applied Statistics for Network Biology: Methods in Systems Biology*, pages 299–327.

[39] UCEPROTECT Network. `http://www.uceprotect.net/`.

[40] US-CERT. Indicators associated with WannaCry ransomware. `https://www.us-cert.gov/ncas/alerts/TA17-132A`.

[41] US-CERT. Petya ransomware. `https://www.us-cert.gov/ncas/alerts/TA17-181A`.

[42] WPBL: Weighted private block list. `http://www.wpbl.info/`.

[43] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.

[44] J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156. IEEE, 2013.

[45] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *Internet Measurement Conference*, pages 15–27, 2009.