Toward an Evidence-based Design for Reactive Security Policies and Mechanisms

Omer Katz
Technion

Benjamin Livshits
Microsoft Research

***Abstract*—As malware, exploits, and cyber-attacks advance over time, so does the mitigation techniques available to the user. However, while attackers often abandon one form of exploitation in favor of a more lucrative one, mitigation techniques are rarely abandoned. Mitigations are rarely retired or disabled since proving they have outlived their usefulness is often impossible. As a result, performance overheads, maintenance costs, and false positive rates induced by the different mitigations accumulate, culminating in an outdated, inefficient, and costly security solution.**

We advocate for a new kind of tunable framework on which to base security mechanisms. This new framework enables a more reactive approach to security allowing us to optimize the deployment of security mechanisms based on the current state of attacks. Based on actual evidence of exploitation collected from the field, our framework can choose which mechanisms to enable/disable so that we can minimize the overall costs and false positive rates while maintaining a satisfactory level of security in the system.

We use real-world Snort signatures to simulate the benefits of reactively disabling signatures when no evidence of exploitation is observed and compare them to the costs of the current state of deployment. Additionally, we evaluate the responsiveness of our framework and show that in case disabling a security mechanism triggers a reappearance of an attack we can respond in time to prevent mass exploitation.

Through a series of large-scale simulations that use integer linear and Bayesian solvers, we discover that our responsive strategy is both computationally affordable and results in significant reductions in false positives, at the cost of introducing a moderate number of false negatives. Through measurements performed in the context of large-scale simulations we find that the time to find the optimal sampling strategy is mere seconds for the non-overlap case and under 2.5 minutes in 98% of overlap cases. The reduction in the number of false positives is significant (about 9.2 million removed over traces that are about 9 years long). The reduction is false positive rates in about 20%.

I. Introduction

Much of the focus in the security community in the last several decades has been on discovering, preventing, and patching vulnerabilities. While both new vulnerability classes and new vulnerabilities are discovered seemingly every day, the exploitation landscape often remains murky. For example, despite buffer overruns, cross-site scripting (XSS), and SQL injection attacks (SQLIA) being heralded as the vulnerabilities of the decade [46], there is precious little published evidence of how commonly *exploited* XSS or SQLIA might be in practice; of course, there is a number of studies [31] on how *vulnerability trends* change over time. One of the studies we present in this paper suggests that, for example, XSS *exploitation* is not nearly as common as would be suggested by the daily stream of discovered *vulnerabilities*[1].

Changing exploitation landscape: The security industry produces regular reports that generally presents a growing number of vulnerabilities that are available, some in widely-deployed software. As we argue in this paper, it is exceedingly tempting to misinterpret this as a growth trend in the number of actual exploits. However, the evidence for the latter is scant at best. Due to a number of defense-in-depth style measures over the last decade, including stack canaries, ALSR, XRF tokens, automatic data sanitization against XSS and a number of others, practical exploitation on a mass scale now requires an increasingly sophisticated attacker. We see this in the consolidation trends of the last five years. For example, individual drive-by attacks have largely been replaced by exploit kits [33], [16], [51]. In practice, mass-scale attacks generally appear to be driven by a combination of two factors: 1) ease of exploitation, and 2) whether attacks are consistently monetizable. This is clearly different from targeted attacks and APTs where the upside of a single successful exploitation attempt may be quite significant to the attacker.

Given the growing scarcity in *exploitable* vulnerabilities, there is some recent evidence that attackers attempt to take advantage of publicly disclosed attacks right after their announcement, while the window of vulnerability is still open and many end-users are still unpatched; Bilge *et al.* [12] report an increase of *5 orders of magnitude* in attack volume following public disclosures of zero-day attacks. The situation described above leads to a long tail of attacks — a period of time when attacks are still possible but are increasingly rare. It is tempting to keep the detection mechanism on during the long tail. However, it is debatable whether that is a good strategy, given the downsides. We argue that the usual human behavior in light of the rapidly-changing landscape is inherently *reactive*, however, often not reactive enough.

A. Mounting Costs of Security Mechanisms Over Time

One of the challenges of security mechanisms is that their various costs can easily mount if unchecked over time.

- **Technical debt.** There is a problem of technical debt associated with *maintaining* existing security mechanisms as attack volume diminishes. This is considered to be

[1]Found at https://www.openbugbounty.org/ or xssed.org.

a growing problem in the machine learning community [50], but is not often cited as an issue in security and privacy. For example, Kerschbaumer [32] reports that only *modernizing* the CSP (content security policy) implementation of the Firefox browser took over 125,000 lines of code over a period or 20 months by a dedicated developer. Clearly, not every project has these kinds of resources. When considered over a long period of time, the technical debt accumulates to a point that the software maker can no longer deal with the maintenance issues or can only do so at the expense of introducing new defense strategies.

- **Performance overhead.** Our studies of AV scanning costs in Section II-C show that while the IO overhead from opening files, etc. can be large, the cost of AV scanning can increase quite significantly as more signatures are added to the signature set. When many solutions are applied within a piece of software, their overheads can be additive, even relatively affordable mechanisms such as stack canaries [15] and ALSR [41]. There is a growing body of evidence that security mechanisms that incur an overhead of 10% or more do not tend to get widely deployed [55]. However, clearly several low-overhead solutions one on top of another can easily exceed the 10% mark.
- **False positives.** FPs have nontrivial security implications [47], [52]. According to a recent Damballa report [43], "The average cost of time wasted responding to inaccurate and erroneous intelligence can average $1.27 million annually."
- **Infrastructure.** It is difficult to estimate the cost of backend maintenance *and* sending security updates. These may include keeping up a VM-based platform for anyzing malware, testing signatures, running a honeymonkey backend for a web crawler, etc. [45], [44].

We argue that as a result of the factors above, over a long period of time, we cannot afford a situation in which we only *add* security mechanisms because of the issue of mounting technical debt and maintenance costs. This is akin to performing a DOS attack against oneself; in the limit, the end-user would not be able to do much useful work because of the overhead and false positive burden of existing software.

Reluctance to disable: At the same time, actively *removing* a security mechanism is tricky, if only from the standpoint of the associated PR. In fact, we are not aware of a recent major security mechanism that has been officially disabled, although, of course, it is possible to deploy effectively a null policy that will have the same effect. One of the obvious downsides of tuning down any security mechanism is that the recall decreases as well. This is used as a counter-argument in terms of lowering any of the shields that might be in place. However, it is important to realize that when tuning down a specific mechanism is driven by representative measurements of how commonly it is encountered in the wild, this is a good strategy for mass attacks.

When it comes to targeted attacks, the challenge is that they are likely to be able to overcome most existing defenses, as we have seen from Pown2own competition, XSS filter bypasses [58], [30], [9], etc. We hypothesize that sophisticated targeted attacks are likely not to be particularly affected by existing defenses. However, reducing the level of defense may invite new waves of mass attacks, which would be mitigated by upping the level of enforcement once again. It is of course always possible to turn up the defenses on-demand. It is also true that some attacks may go through a period of downtime, to be followed by a revival (which can be triggered by the removal of security mechanisms). As a result, entirely taking out a security mechanism might be ill-advised.

B. Toward Tunable Security Mechanisms

In recent years we have seen growing evidence of the fact that vulnerability statistics and exploit statistics are at odds. In fact, Nayak *et al.* [39] report an increase in reported vulnerabilities in the last several years, while the amount of exploitation goes down. Furthermore, only a fraction of vulnerabilities (about 35%) actually gets exploited in a practical sense. Furthermore, often the vulnerability severity rankings are misleading, as they does not necessarily correlate with the attractiveness or practicality of exploitation [5]. Barth *et al.* [8] advocate a reactive approach to security where previous attack patterns are used to decide how to spend the defender's finite budget. In this paper, we agree with the advantages of reactive security. However, we look at widely deployed security mechanisms, where the potential, for example, of a single false positive is amplified by the potentially vast installation base. A good example of such a system is an anti-virus (AV) engine or an IDS. The reactive approach to security is also supported by the number of zero-days that are observed in the wild and reported by Bilge *et al.* [12].

We propose an alternative: we can have a tunable mechanism, a strategy that allows the defender to vary the level of application based perhaps on an internal or external set of conditions. For example, a centralized server can notify the client that certain categories of attacks are no longer in the wild, causing the client to reduce their sampling rates when it comes to suspicious traffic. This is not unlike what is already done in anti-virus engines — the set of active anti-virus signatures is what defines what gets examined. Signatures are retired when the threats they are designed to look for become less prevalent. For this to work, there needs to be a greater level of independence between the security mechanism and the policy, the kind of separation that is already considered a major design principle [4], [35].

In many ways, our proposal is aligned with practical security enforcement practices of adjusting the sensitivity levels for detectors depending on the FP-averseness of the environment in which the detector is deployed. The Insight reputation-based detector from Symantec allows the user to do just that [53].

Our goal here, of course, is to reduce all the factors listed above, i.e. the false positive rate, the performance overhead,

and the amount of possible back-end processing that is involved.

C. Contributions

Our paper makes the following contributions:

- **Tunable.** We point out that today's approach to proactive security leads to inflexible and bloated solutions over time. We instead advocate a notion of tunable security design, which allows flexible and fine-grained policy adjustments on top of existing security enforcement mechanisms. This way, the protection level is tuned to match the current threat landscape and not either the worst-case scenario or what that landscape might have been in the past.
- **0/1.** For a collection of mechanisms that can be turned on or off independently, we propose a strategy for choosing which mechanisms to enable for the optimal combination of true positives, false positives, and performance overheads.
- **Sampling-based approach.** We formalize the problem of optimal adjustment for a mechanism that includes an ensemble of classifiers, which, by adjusting the sampling rates produces the optimal combination of true positives, false positives, and performance overheads.
- **Simulation.** Using a simulation based on a history of Snort signature updates over a period of about 9 years, we show that we can adjust the sampling rates within a window of minutes. This means that we can rapidly react to a fast-changing exploit landscape.

D. Paper Organization

The rest of the paper is organized as follows. Section II gives an overview of the exploitation landscape. Section III defines an optimization problem that improves the true positive rates and reduces the false positive rate and enforcement costs, while favoring higher-severity warnings. Section IV describes our experimental evaluation. Finally, Sections V and VI describe related work and conclude.

II. Background

When it comes to malware detection, anti-virus software (or AV, for short), has long been the first line of defense. However, for almost as long as AV engines have been around, they have been recognized to be far from perfect, in terms of their false negative rates, false positive rates, and, lastly, in terms of performance [34], [17]. When it comes to false negatives, the recall of AV engines is infamously low [22]. According to a whitepaper by Tenable [57], AV detection rates range from about 5% to about 35%. Even ignoring the possibility of zero-days [12], [11], AV vendors who manually generate new signatures frequently have vulnerability windows of 2–8 weeks, which indicates that given the speed with which malware is manufactured or mutated by attackers, defenders have a hard time keeping up, leading to false negatives.

Clearly, the choice of the signature database plays a decisive role in the success of the the AV solution. To illustrate this point, consider a company, SaneSecurity[2], which promises to deliver detection rates of up to 90% by using the free open-source ClamAV detection engine and their own carefully curated and frequently updated database of signatures. For example, as of August 2016, they claim a detection rate of 97.11% vs only 13.82% for out-of-the-box ClamAV, using a database of little over 4,000 signatures vs. almost 250,000 for ClamAV.

Several researchers have proposed automatically generating new signatures automatically [40], [42], [24], [49], [61]. Signature *addition* seems to largely remain a manual process, supplemented with testing potential AV signatures against known deployments, often within virtual machines.

We hypothesize that there is a reluctance to remove or disable older signatures, leading to an unnecessary scanning burden on the AV engine. While the issue of false negatives is generally known to industry insiders, at the same time, the issue of false *positives* receives much more negative press; so, delayed signature removal runs the risk of unnecessarily triggering false positives, which are supremely costly to the AV vendor, as evidenced by studies by AV-Comparatives [47][3].

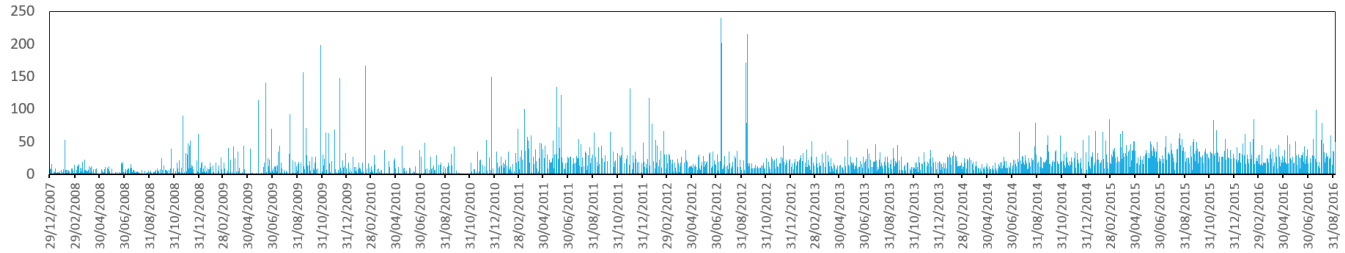
A. The Changing Attack Landscape

When the attacks for which some mitigation mechanism was designed are no longer observed in the wild, it might seem very alluring to remove said mechanism. However, in most real world scenarios, we are not able to fully retire mitigation techniques. Before disabling some mitigation mechanism, one should examine the reason these attacks are no longer being observed and also whether this is simply due to the observational mechanism and data collection approach being faulty.

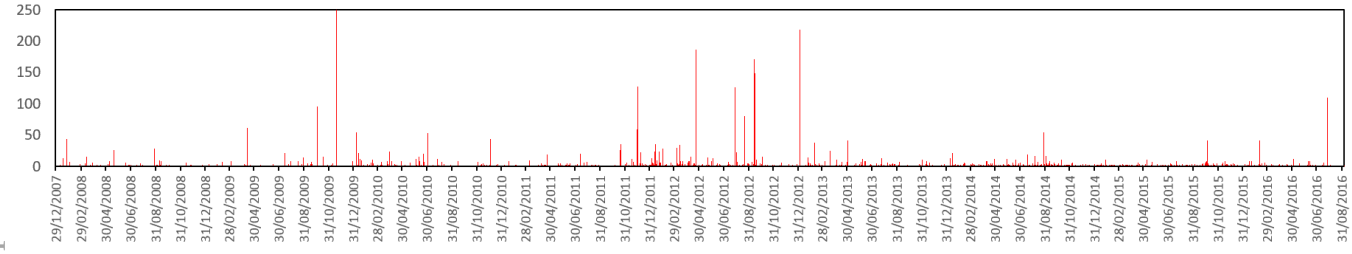
Today, large-scale exploitation is often run as a business, meaning it is driven largely, but not entirely, by economic forces. The lack of observed attacks might simply be associated with an increase in difficulty in monetizing the attack. Given that the attacker is aiming to profit from the attack, if the cost of mounting a successful attack is too high compared to either the possible gains or alternative attack vectors, the attacker will most likely opt not to execute it as it is no longer cost-effective. We see these forces in practice as the attack landscape changes, with newer attacks such as ransomware becoming increasingly popular in the last several years and older attacks leading to the theft of account credentials becoming less common because of two-factor authentication, geo-locating the user, etc. To summarize, we identify two common cases where monetizing becomes hard

- **No longer profitable.** The first is the result of causes other than the mitigation mechanism. For example, when clients are no longer interested in the possible product of the attack or if there are other security mechanisms in place that prohibit the usefulness of the attack's outcome. In such cases, removing the mitigation mechanism in question will most likely not have a practical negative

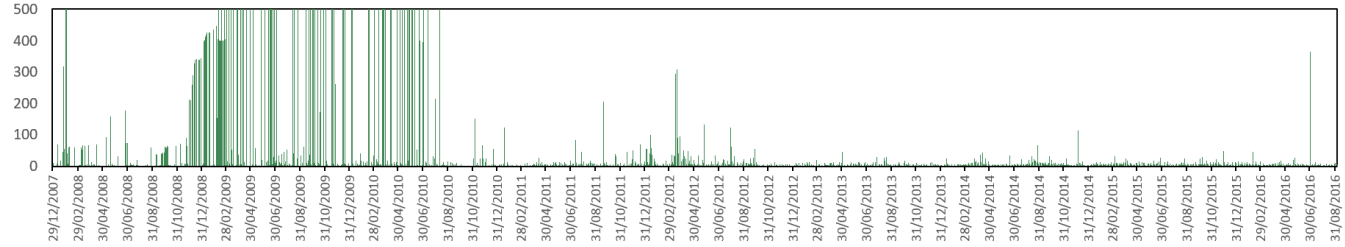
[2] http://sanesecurity.com.
[3] http://www.av-comparatives.org



(a) Additions of signatures in the Snort emerging threats database.



(b) Removals of signatures in the Snort emerging threats database.



(c) Updates of signatures in the Snort emerging threats database.

Fig. 1: Dynamics of Snort signatures between 12/30/2007 and 9/6/2016.

```
2023020 - ET TROJAN ProjectSauron Remsec DNS Lookup (rapidcomments.com) (trojan.rules)
2023021 - ET TROJAN ProjectSauron Remsec DNS Lookup (bikessport.com) (trojan.rules)
2023022 - ET TROJAN ProjectSauron Remsec DNS Lookup (myhomemusic.com) (trojan.rules)
2023023 - ET TROJAN ProjectSauron Remsec DNS Lookup (flowershop22.110mb.com) (trojan.rules)
2023024 - ET TROJAN ProjectSauron Remsec DNS Lookup(wildhorses.awardspace.info) (trojan.rules)
2023025 - ET TROJAN ProjectSauron Remsec DNS Lookup (asrgd-uz.weedns.com) (trojan.rules)
2023026 - ET TROJAN ProjectSauron Remsec DNS Lookup (sx4-ws42.yi.org)(trojan.rules)
2023027 - ET TROJAN ProjectSauron Remsec DNS Lookup (we.q.tcow.eu)(trojan.rules)
```

(a) ProjectSauron malware (`http://http://bit.ly/2eX1O9h`).

```
2003182 || ET TROJAN Prg Trojan v0.1-v0.3 Data Upload || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2003183 || ET TROJAN Prg Trojan Server Reply || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2003184 || ET TROJAN Prg Trojan v0.1 Binary In Transit || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2003185 || ET TROJAN Prg Trojan v0.2 Binary In Transit || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2003186 || ET TROJAN Prg Trojan v0.3 Binary In Transit || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2007688 || ET TROJAN Prg Trojan HTTP POST v1 || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
2007724 || ET TROJAN Prg Trojan HTTP POST version 2 || url,www.securescience.net/FILES/securescience/10378/pubMalwareCaseStudy.pdf
```

(b) Zeus (Prg) malware (`http://bit.ly/2bIS3hk`).

Fig. 2: Connecting signatures to known malware

effect on the system since the attack remains not cost-effective.

- **Effective mitigation.** The other case is when the attack is not cost-effective due to difficulties imposed by the mitigation mechanism. In such cases, removing the mechanism will result in an increase in the cost-effectiveness of the attacks it was aimed to prevent. This might result in the reemergence of such attacks.

By sampling the relative frequency of attacks of a particular kind, we cannot always determine which case we are currently faced with. It may be a combination of these two factors as well.

We therefore suggest an alternative that acts as a middle ground by introducing sampling rates for all mitigation mechanisms. In the first case, the mitigation mechanism is no longer needed, therefore adding a sampling rate will reduce the security of the system, but will provide fewer benefits than a complete removal. On the other hand, in the second case, the security of the system is somewhat lowered, but the statistical nature of the sampling rate maintains some deterrence against attackers.

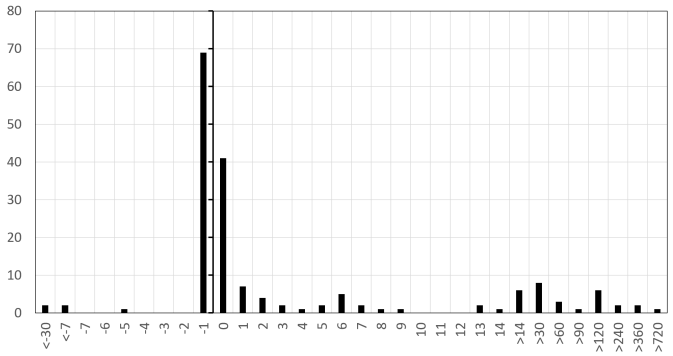


Fig. 3: How many days after (or before) the CVE announcement we observe an EMERGING THREATS signature being added. Positive numbers mean that there is a delay, whereas negative numbers indicate a signature added *before* the CVE.

B. *Snort Signatures from* EMERGING THREATS

To test some of these educated guesses, we have performed an in-depth study of Snort signatures. Focusing on the dynamics of signature addition and removal more specifically, we have mined the database of Snort signatures, starting on 12/30/2007 and ending on 9/6/2016. Daily updates to the Snort signature database are distributed through the EMERGING THREATS mailing list archived at https://lists.emergingthreats.net, which we used to determine which signatures, if any were 1) added, 2) removed, or 3) modified every single day. The results of exploring the dataset we obtained by crawling the mailing list archive are presented in figure 1.

Below we give several representative examples of signature addition, update, and removal.

Example 1 Signature addition. Figure 2a shows the addition of new signatures in response to observations of the malware known as ProjectSauron [23] in the wild. The connection between EMERGING THREATS signatures and the malware they are designed to protect against is evident from the signature description.

Similarly, Figure 2b shows the connection between EMERGING THREATS signatures and variants for the banking Trojan called Zeus [10] (also known as Prg). □

Example 2 Signature update. Figure 4 shows an example of a typical mailing list exchange leading up to a signature change. However, the promised update does not get added to the signature database until a later date, 02/04/2011. It is not entirely clear why.

Similarly, in the case of signature 2011124, we see a false positive report about traffic on port 110 on 04/04/2016, which receives a response from the maintainers within two days:

```
we get quite a lot of false positives with this one due to
the POP3 protocol on port 110, it would be great if port 110
or more generally POP3 traffic could be excluded from this rule
-- JohnNaggets - 2016-04-02
Thanks, we'll get this out today!
-- DarienH - 2016-04-04
```

The maintainers added port 110, resulting in this signature revision 19:

```
alert ftp $HOME_NET ![21,25,110,119,139,445,465,475,587,902,1433,2525] ->
    any any (msg:"ET MALWARE Suspicious FTP 220 Banner on Local Server (spambot)";
```

```
> We have 2010148 already:
> content:"Content-Disposition|3A| attachment|3b|"; nocase;
> content:"filename"; within:100; content:"DHL_"; nocase;
> within:50;
> pcre:"/filename\s*=\s*\"DHL_(Label_|document_|
> package_label_|print_label_).{5,7}\.zip/mi";
>
> I'll modify to fit the new style. The old is gone!
>
> Thanks Jason!
>
> Matt
>
> On Nov 3, 2010, at 10:35 AM, Weir, Jason wrote:
>
> > Seeing these as inbound smtp attachments
> >
> > DHL_label_id.Nr21964.zip
> > DHL_label_id.Nr48305.zip
> > DHL_label_id.Nr3139.zip
> > DHL_label_id.Nr15544.zip
> > DHL_label_id.Nr7085.zip
> >
> > How about this for current events
> >
> > alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (msg:"ET
> > CURRENT_EVENTS DHL Spam Inbound"; flow:established,to_server;
> > content:"Content-Disposition|3a| attachment|3b|"; nocase;
> > content:"filename=|22|DHL_label_id."; nocase;
> > pcre:"/filename=\x22DHL_label_id\.Nr[0-9]{4,5}\.ZIP\x22/i";
> > classtype:trojan-activity; sid:xxxxxxx; rev:0;)
```

Fig. 4: Updating a signature based on customer feedback.

```
    flow:from_server,established,only_stream; content:"220 ";
    depth:4; content:!"SMTP"; within:20;
    reference:url,doc.emergingthreats.net/2011124;
    classtype:non-standard-protocol; sid:2011124; rev:19;)
```

the same day they responded. □

CVE Additions: Evaluation of the Delay: Another question to consider is how fast signatures for known threats are added after the threats are discovered or publicly announced. To estimate that, we have correlated 176 CVEs to 40,884EMERGING THREATS signatures. This process usually involves analyzing the comments embedded in the signature to find CVE references (for example, `reference:cve,2003-0533`). We have plotted a histogram that shows the *delay* in days between the CVE (according to the NIST NVD database) and the signature introduction date. The results are shown in the form of a histogram in Figure 3. As we can see, many signatures are created and added the same day the CVE is disclosed. Sadly, in quite a significant percentage of cases, signatures are added two weeks and more *after* the CVE release date. What is perhaps most surprising is that many signatures are created quite a bit *before* the disclosure, in many cases the day before, and in some cases over a month prior to it. This must be due to other information sources that lead to signature generation.

C. *AV Scanning Costs*

To demonstrate our claims of the inefficiency and mounting costs of maintaining a large amount of outdated security mechanism, we turn to ClamAV [1]. We installed the latest version of the ClamAV engine (0.99.2) and used it to scan a single file of 248 MB. To understand how the running time is affected by the size of the signature set, we ran the scan several times using different sized subsets of the ClamAV

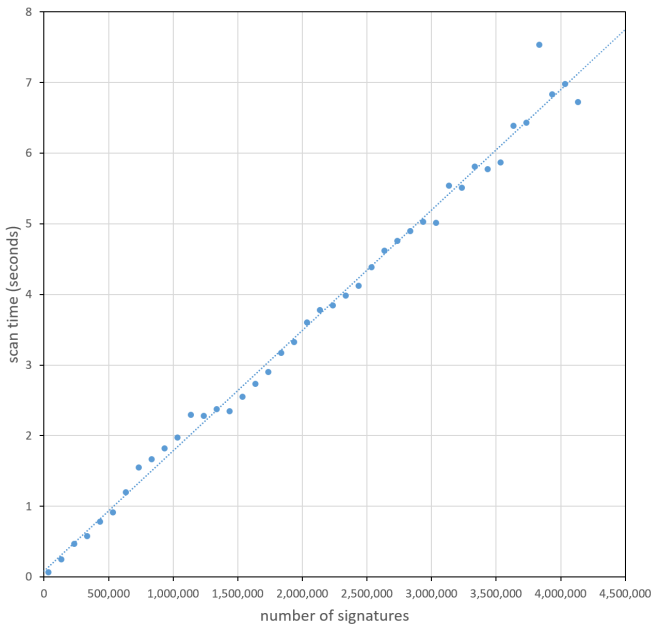


Fig. 5: Scanning time for a single 248 MB file using ClamAV version 0.99.2 as a function of the signature dataset size. The dashed line is a linear fit to the measurements.

signature dataset. Figure 5 shows the obtained scan times, in seconds. The figure clearly demonstrates a linear correlation between the signature dataset size and the scan time, which is represented by the dashed trend line.

This data clearly demonstrates the hidden cost over time of adding signatures and rarely removing them. This factor together with false positives argues for removing signatures more aggressively.

III. Optimally Setting Sampling Rates

In this section we set the stage for a general apporach to selecting sampling rates in response to changes to the data. We evaluate these ideas with practical multi-year traces in Section IV. We start with a model in which we have a set of classifiers at our disposal and we need to assign a sampling rate to each of them, so as to match our optimization goals. These goals include higher true positive rates, lower false positive rates, and lower overheads. We formalize this as problem of selecting a bit-vector α, which indicates the sampling rate for each classifier.

A. Active Classifier Set: a Formalization

Our assumption is that our classifiers send some portion of the samples they flagged as malicious for further analysis. This matches how security products, such as those from Symantec, use user machines for in-field monitoring of emerging threats. In the context of a large-scale deployment, this will result in a large, frequently updated dataset, which consists solely of true positive and false positive samples. We can use this dataset to evaluate the average true positive, false positive, true negative, and false negative rates, induced by each sampling rate for our classifiers. Specifically, our aim is to choose a sampling bit-vector $\bar{\alpha}$ that will keep the true positive and true negatives above some threshold, while keeping the false positive rate, false negatives, and performance costs below some maximum acceptable values. We found this formulation to be most useful in our evaluation.

Constraints: Formally, these goals can be specified as a set of inequalities, one for each threshold:

$$TP(\bar{\alpha}) \geq X_p \tag{1a}$$

$$TN(\bar{\alpha}) \geq X_n \tag{1b}$$

$$FP(\bar{\alpha}) \leq Y_p \tag{1c}$$

$$FN(\bar{\alpha}) \leq Y_n \tag{1d}$$

$$Cost(\bar{\alpha}) \leq Z \tag{1e}$$

Parametrization: Given a dataset D and a set of classifiers C we define the following parametrization:

- D_i is the i^{th} entry in the dataset and C_j is the j^{th} classifier;
- $G \in (0/1)^{|D|}$, such that G_i is 1 iff D_i is a malicious entry in the ground-truth;
- $R \in (0/1)^{|D|\times|C|}$, such that $R_{i,j}$ is 1 if D_i is classified as malicious by C_j or 0 otherwise;
- $P \in \mathbb{R}^{|C|}$, such that P_j is the average cost of classifying an entry from the dataset using C_j;
- $\alpha \in [0,1]^{|C|}$, such that α_j is the sampling rate for classifier c_j.

For each set sampling rate α we can compute the average cost of executing the entire set of classifiers on an entry from the dataset as:

$$Cost(\alpha) = P^T \cdot \bar{\alpha} \tag{2}$$

Optimization: To evaluate the true positive and false positive rates induced by a sampling rate α, we first need to evaluate the probability that an entry will be classified as malicious. Given a constant R, this probability can be expressed as:

$$Pr_i(\alpha) = 1 - \Pi_{j=0}^{|C|}(1 - R_{i,j} \cdot \alpha_j) \tag{3}$$

Based on this probability, we can express the true/false-positive rates as:

$$TP(\bar{\alpha}) = \frac{\Sigma_{i=0}^{|D|}(G_i \cdot Pr_i(\alpha))}{\Sigma_{i=1}^{|D|}(G_i)} \tag{4a}$$

$$FP(\bar{\alpha}) = \frac{\Sigma_{i=0}^{|D|}((1-G_i) \cdot Pr_i(\alpha))}{\Sigma_{i=1}^{|D|}(G_i)} \tag{4b}$$

$$TN(\bar{\alpha}) = \frac{\Sigma_{i=0}^{|D|}((1-G_i) \cdot (1-Pr_i(\alpha)))}{\Sigma_{i=1}^{|D|}(1-G_i)} \tag{4c}$$

$$FN(\bar{\alpha}) = \frac{\Sigma_{i=0}^{|D|}((G_i \cdot (1-Pr_i(\alpha)))}{\Sigma_{i=1}^{|D|}(1-G_i)} \tag{4d}$$

In practice, not all suggested goals are always necessary and not all goals are always meaningful. Finding the optimal

sampling rate usually depends on the setting for which it is needed. Next we discuss a few hypothetical scenarios and which approaches might best suit them.

Prioritized objectives: When the user can state that one objective is more important than others, a multi-leveled optimization goal can be used. In such a solution, the objective with the highest priority is optimized first. In case there is more than a single possible solution, the second objective is used to choose between them, and so on.

We note that in our scenario it is extremely unlikely that one can reduce the sampling rate of a classifier without affecting the true-positive and false-positive rates. As a result, using strict objectives, such as maximize true-positives, would result in a single solution, often enabling all classifiers completely (or disabling all, depending the chosen objective). Therefore it is recommended to phrase the objectives as "maintain X% of true-positives", so that some flexibility remains.

Budget-aware objectives: Often when assessing the effect a security mechanism has on a company's budget, a cost is assigned to each false positive and each false negative produced by the mechanism. These assessments can be used to minimize the total budgetary effect of the mechanism and expected expenses. Assuming $Cost_{FN}$ and $Cost_{FP}$ are the costs of false negatives and false positives respectively, we can express the expected expenses as:

$$Expenses(\alpha) = Cost_{FN} \cdot FN(\alpha) + Cost_{FP} \cdot FP(\alpha) \quad (5)$$

Using this formulation we can:

- Define a budget, $Expenses(\alpha) \leq BUDGET$, as a strict requirement from any sampling rate.
- Define our problem as a standard optimization problem with the objective $minimizeExpenses(\alpha)$.

Balancing true positives and false positives: In this scenario, the sampling rate optimization problem can be translated to a standard classifier optimization setting. Under this translation, our true-positive rate is equivalent to the classifier's precision while the false-positive rate becomes the recall.

In such a case a ROC curve induced by different sampling rates can be used to select the best rate. Taking some inspiration from the well-known $F1$-score, a similar score, $F1_{sr}$, expressed in formula 6 can be used to transform our problem to a single-objective optimization problem.

$$F1_{sr} = 2 \cdot \frac{TP \cdot FP}{TP + FP} \quad (6)$$

For efficiency, we split the process of classifier sampling rate optimization into two steps. Real-world data often contains classifier overlap, that is, samples that are flagged by more than one classifier. We split our dataset into batches based on the classifier overlap, so that the samples in each batch are flagged by the same set of classifiers. Each batch is associated with true positive and false positive counts. The first step consists of choosing the batches that are cost-effective.

B. 0/1 Sampling Using Linear Optimizations

Based on the desired optimization objective and the estimated cost ratio between false negatives and false positives (if applicable), we proceed to define the problem of finding the optimal subset of sample batches as a *linear programming* problem. At this stage, since each batch is determined by a specific classifier overlap, there is no overlap between the batches. Therefore, the computation of the true/false positives/negatives becomes a simple summation of the associated true/false positive counts. For example, the total true positive count is the sum of true positives associated with batches that are determined as enabled (meaning they should be sampled) and the total false negative count is the sum of the true positive counts of disabled batches.

To encode this problem, we assign each batch b_i with a boolean variable v_i, representing whether or not the batch should be active. We then encode the optimization goal using these variables and the associated counts. We use a linear programming solver called Pulp [3], which finds an assignment to $\bar{v} = \{v_1, v_2, ...\}$ that optimizes the optimization objective. The output of this step is a division of the samples into enabled, meaning the classifier should sample them, and disabled samples.

When the dataset contains *no classifier overlap*, meaning each sample is sampled by exactly one classifier, the output of the first step can be used as the classifier sampling rates. In this setting, the batches essentially correspond to single classifiers and therefore disabled batches correspond to classifiers that are deemed not cost-effective according to the optimization objective. The optimal solution in this case would be to fully enable all cost-effective classifiers and fully disable the rest.

C. Inferring Sampling Rates using Factor Graphs

In the second step of our solution, given a set of enabled samples and a set of disable samples as described in Section III-B, we infer sampling rates for all classifiers that will induce the desired separation.

The key insight we use to infer the classifier sampling rates is to express our problem in the form of *factor graphs* [36]. Factor graphs are probabilistic graphical models composed of two kinds of nodes: variables and factors. A variable can be either an evidence/observation variable, meaning it's value is set, or a query variable, whose value needs to be inferred. A factor is a special node that define the relationships between variables. For example, given variable A and B, a possible factor connecting them could be $A \rightarrow B$.

Example 3 Consider the example factor graph in Figure 6. The graph consists of 8 variables: 3 named C_1–C_3 representing 3 different classifiers and 5 variables named S_1-S_5 representing samples. These variable are connected using 5 factors, F_1–F_5, such that $F_i(\bar{C}, S_i) = \vee\bar{C} \rightarrow S_i$, where $\bar{C}$ is the set of classifiers with edges entering the factor.

The variables S_i are treated as observations, meaning their value is set, and the variables C_i are treated as query variables. The inference algorithm has to choose at which probability is

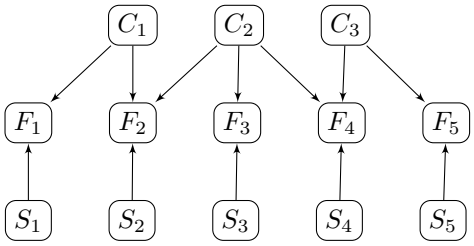


Fig. 6: Example factor graph, containing three classifiers $C_1 \dots C_3$ and five samples $S_1 \dots S_5$. The structure of the factor graph determines the overlap among the classifiers.

each C_i set to `true`, such that the factors satisfy the observations. If we set all observations S_i to `true`, the inference of the factor graph returns the trivial solution of always setting all C_i to `true` (meaning C_i is `true` with probability 1.0).

When we set some S_i to `false`, the inference algorithm is able to provide more elaborate answers. For example, setting S_4 to false, results in probability 1.0 for C_1 and probability 0.5 to both C_2 and C_3. □

Given the sets of enabled and disabled samples from the previous step, we translate the problem to a factor graph as follows:

1) For each classifier i we define a query variable C_i;
2) for each sample j we define an observation variable S_j and a factor F_j;
3) if sample j was set to be enable we set S_j to `true`, otherwise to `false`;
4) we connect each S_j to its corresponding F_j;
5) for every pair of classifier and sample (i, j), if classifier i flags sample j we connect C_i to F_j.

Using this construction, we get a factor graph similar in structure to the graph in figure 6. The inferred probabilities for the query variables C_i are used as the sampling rates for the corresponding classifiers.

To solve the factor graph and infer the probabilities for C_i, we use Microsoft's Infer.NET [2], with `ExpectationPropagation` chosen as the inference algorithm, which worked fastest in our experiments. It took us some effort to find a specific problem formulation that works well with the Infer.NET solver. We evaluate the performance of the solver in Section IV.

D. Discussion

Maintaining the dataset: We intend to build our dataset using samples flagged as malicious by our classifiers. This kind of dataset will naturally grow over time to become very large. Two problems rise from this situation.

The first problem is that after a while most of the dataset will become outdated. While we usually wouldn't want to completely drop old samples, since they still represent possible attacks, we would like to give precedence to newer samples over older ones (which essentially should result in higher sampling rates for current attacks). To facilitate this we can assign a weight to each sample in the dataset. We represent these weights using $W \in [0,1]^{|D|}$ and rewrite the formulas from 4 as:

$$TP(\alpha) = \frac{\Sigma_{i=0}^{|D|}(W_i \cdot G_i \cdot Pr_i(\alpha))}{\Sigma_{i=1}^{|D|}(W_i \cdot G_i)} \tag{7a}$$

$$FP(\alpha) = \frac{\Sigma_{i=0}^{|D|}(W_i \cdot (1-G_i) \cdot Pr_i(\alpha))}{\Sigma_{i=1}^{|D|}(W_i \cdot G_i)} \tag{7b}$$

$$TN(\alpha) = \frac{\Sigma_{i=0}^{|D|}(W_i \cdot (1-G_i) \cdot (1-Pr_i(\alpha)))}{\Sigma_{i=1}^{|D|}(W_i \cdot (1-G_i))} \tag{7c}$$

$$FN(\alpha) = \frac{\Sigma_{i=0}^{|D|}(W_i \cdot G_i \cdot (1-Pr_i(\alpha)))}{\Sigma_{i=1}^{|D|}(W_i \cdot (1-G_i))} \tag{7d}$$

While many different weighting techniques can be used, two examples are:

- Assign weight 0 to all old samples, essentially dropping old samples from the dataset.
- Assign some initial weight w_0 to each new sample and exponentially decrease the weights of all samples after each sampling rate selection.

The second problem stems from the sampling rates themselves. Given 2 classifiers, C_1 and C_2, and their corresponding sampling rates, α_1 and α_2, if α_1 is higher than α_2 the dataset will contain more samples of attacks blocked by C_1 than by C_2. This creates a biased dataset that, in turn, will influence sampling rates selected in the future. This problem can also be addressed using the weights mechanism. One possible approach will be to assign initial weights in reverse ratio to the sampling rates (so that samples matching C_2 will be assigned a higher rate than samples matching C_1). Other viable approaches exist and the most suitable approach should be chosen based on the setting in which the classifiers are used.

Minimum sampling rates: In Section III-A we've defined a sampling rate as $\alpha \in [0,1]^{|C|}$. This definition allows for a complete disable of a classifier by setting it's sampling rate to 0. In practice, since we can never be sure that an attack has completely disappeared from the landscape, it is unlikely that we will want to completely disable a classifier.

A possible approach to address this is by setting a minimal sampling rate for the classifier. Given that the attack for which this classifier was intended is extremely unlikely to be encountered we don't want to apply the classifier to every sample encountered. however since the attack is still possible we should statistically apply the classifier to some samples to maintain some chance of blocking and noticing an attack (if one appears). Given an inferred sampling rate S, the minimal sampling rate can be introduced in many forms, such as

- a lower bound L on the sampling rate assigned to each classifier ($S >= L$);
- a constant value X added to the sampling rate ($S + X$);
- some percentage Y reduced from the non-sampled portion ($S + (1 - S) \cdot Y$).

We note that the minimum sampling rate for each classifier should be proportional to the severity of the attacks for

which it was intended. If the impact of a successful attack is minuscule, we may set a lower minimum sampling rate because even if we miss the attack the consequences are not severe. However, if the impact is drastic, meaning the severity of the attack is high, then we should set a higher minimum sampling rate as a precaution.

We can formalize the notion of minimal sampling rates as $MinSR \in [0,1]^{|C|}$, which is based on some severity mapping $S \in \mathbb{N}^{|C|}$ (such that S_j is the severity of the attacks for which classifier C_j was intended), and use $MinSR_j$ as either L,X, or Y from the examples above.

IV. Experimental Evaluation

In this section we first describe our simulation design and then discuss both how much our approach helps in terms of achieving optimization objectives such as reducing false positives, and how long it takes to solve the optimization problems on a daily or weekly basis.

A. Simulation Design

To evaluate the benefits of our approach we performed several simulations simulating real world conditions. We design our simulation with a goal of mimicking real-world anti-virus activity. For the purposes of our simulation, we collected detailed information about Snort signature activity summaries from 12/30/2007 until 9/6/2016, entailing signature additions, updates and removals, as shown in Figure 1. In total, we've collected information regarding 40,884 signatures, each of which we use as a classifier in our simulation.

Generating simulated malware traffic: We generate malware traffic traces (observed true positive and false positive samples) based on the collected Snort signature information. We assume that

- each signature was introduced to counteract some specific malware;
- for each malware, some other active signatures might unintentionally flag that malware even though those signatures were not aimed for that specific malware (resulting in classifier overlap), and
- signature updates are aimed to address some false negatives, resulting in increased true positive and false positive observations.

To simulate the decline of a specific type of malware over time, we use a power law decay curve, which we calibrate to fit the lifespan of the collected signatures. We filter out signatures which were added or removed outside of our sampling period, for which we can't determine a life span, and eliminate short-lived signatures (less than 7 days), leaving 3,029 signatures, which we use in the simulation. Figure 7 shows the number of active signatures for each day of our simulation.

We note that in many real-world cases, a signature is introduced only a few days after a malware appears in the wild and is removed *at least* a few days after the relevant malware disappeared from the attack landscape. We incorporate this insight into the attack model.

We also note that because legitimate traffic does not change, the amount of false positive observations should remain constant as long as the signature is not changed. We therefore only update the false positive traces when we encounter a signature update, in which case we set the number of observations as some percentage, denoted θ, of the true positive observations. We leave θ as a parameter for the simulation. The curve in Figure 8 show an example of the number of true positive and false positive observation for Snort signature 2007705.

Modeling classifier overlap: From the collected signature information, we learn that, while there exists some overlap between signatures, most signatures only flag one kind of malware. To simulate overlap in our generated traces, we randomly choose for each signature with how many other signatures it overlaps. We draw this value from the distribution shown in Figure 9.

Simulation scenario: We aim to simulate a real world usage in our simulation. The scenario we are simulating is when *once every 3 days* our tool is applied to the latest observations and updates the sampling rates for all active signatures. New signatures might still be introduced between sampling rate updates and are set to full sampling until the next update. We believe this to be a reasonable setting that is quite likely to be implemented in practice.

Additionally, under some conditions, Infer.NET's inference algorithm might fail. Such conditions are very rare (inference for only 1.5% of days either failed or timed-out). However, if they occur we allow the simulation to keep using the sampling rates computed on the last update. We believe this to be a reasonable solution that is most likely to be used in practice in case of inference failure.

We defined our optimization goal using a budget-aware objective. We assume a known estimated ratio, denoted as β, between the cost of a false negative and that of a false positive, and phrase the objective as $FP + \beta \cdot FN$. We leave β as a parameter for the simulation. We initialized each malware with an initial true positive count of approximately 500 observations per day.

B. Experimental Setup

To compare different simulation conditions, we ran several simulations, each with a different combination of values for θ and β, both with classifier overlap and without. The simulations were executed on a Linux machine with 64 AMD Opteron(TM) 6376 processors, each operating at 2.3GHz, and 128 GB RAM, running Ubuntu 14.04. Each simulation was assigned the exclusive use of a single core.

C. Precision and Recall Results

By applying the sampling rates computed by our system, one can eliminate part of the false positives previously observed at the expense of losing part of the true positive observations. In Figure 10, we shows the percentage of true positives remaining compared to the percentage of false positives eliminated.

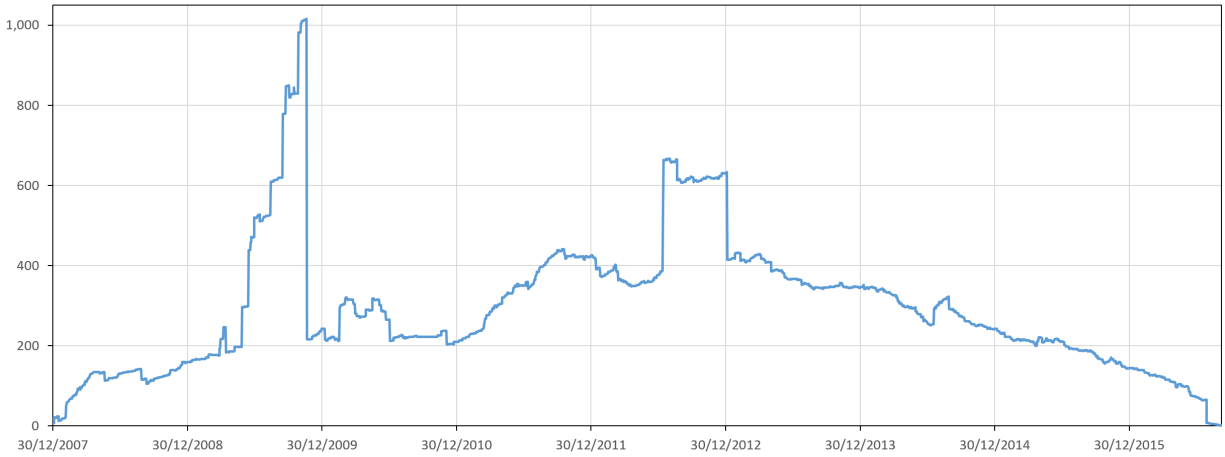


Fig. 7: Number of active signatures in the Snort archive (12/30/2007–9/6/2016). Figure 1 shows some of the update dynamics.

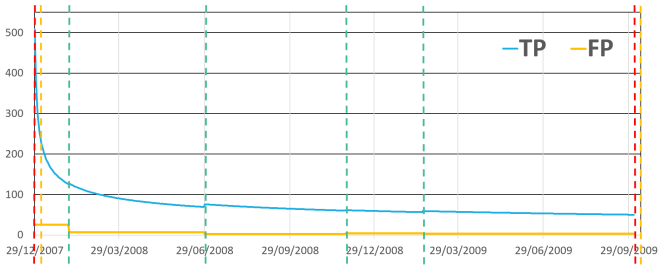


Fig. 8: Modeling the true positive count and the false positive count per day for EMERGING THREATS signature 2007705, assuming power law decay. The dashed lines at the ends of the figure indicate malware appearance, signature introduction, malware disappearance, and signature removal. The dashed lines in the middle of the figure indicate signature updates.

The dashed line across each of the figures symbolizes an equal loss of both false positives and true positives. The area above the dashed line matches settings in which less true positives are lost compared to false positives. This is the area we should strive to be in, since it represents a sampling which is relatively cost-effective. One can clearly see from the figures that, regardless of classifier overlap, all of our simulations reside above the dashed line.

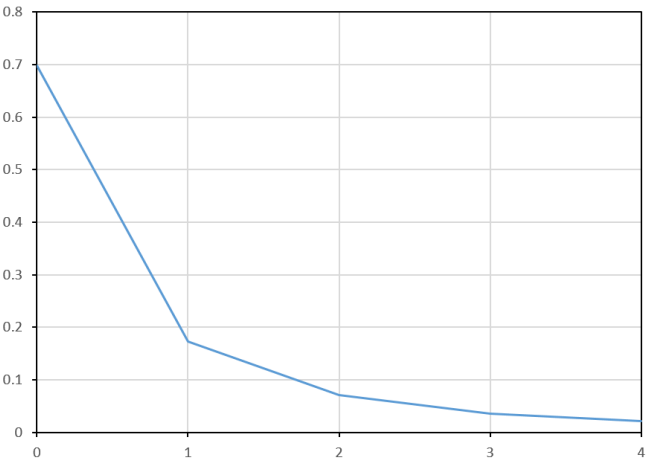


Fig. 9: Probability distribution used to decide amount of overlap for each signature.

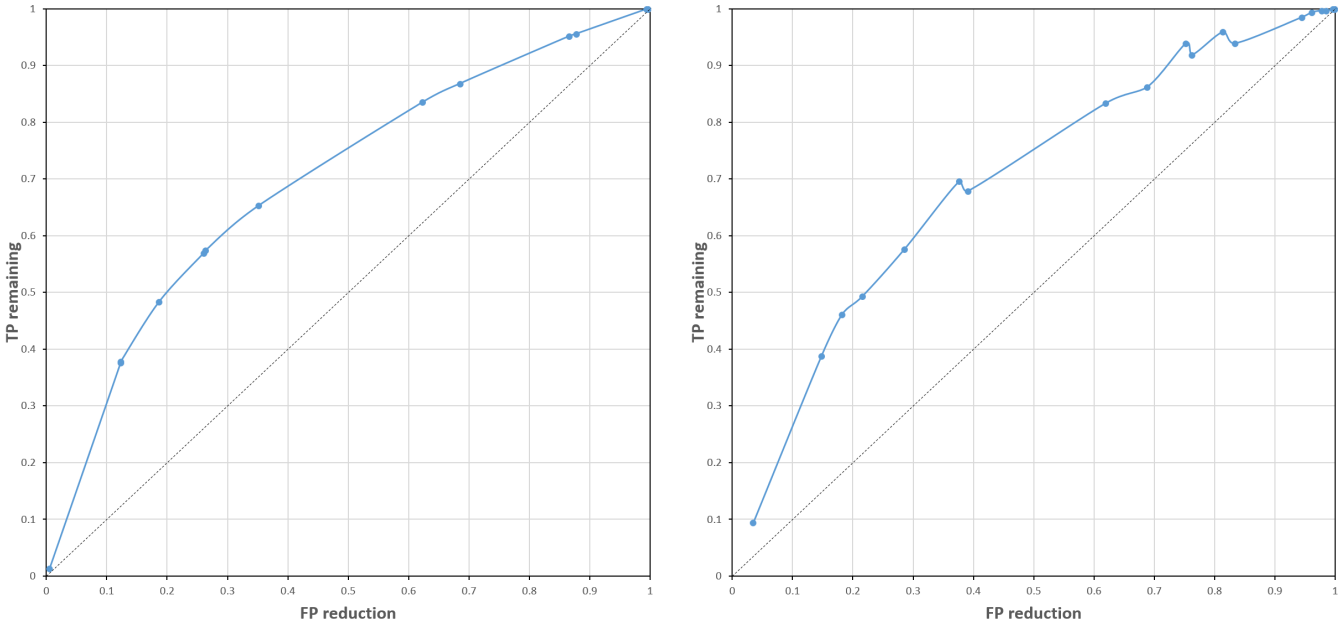
Figures 11 and 12 show the classification precision and recall respectively as a function of θ for different value of β, both with and without classifier overlap. The figures show that, regardless of the overlap, both precision and recall drop when β and θ rise. A rise of θ means there are more false positive observations, which reduces the portion of observed true positives, thus affecting the overall precision and recall. Similarly, a rise of β means that relative cost of a false negative is higher than that of a false positive. Therefore, based on the optimization objective we set in Section IV-A, it is only logical that the system will choose to allow for more false positives, rather than risking a false negative, thus again affecting both precision and recall.

Adapting to the situation: From the aforementioned figures, we learn that the effectiveness of applying sampling rates depends greatly on the operating scenario. In some cases, where for example false negatives are extremely expensive (as might be the case for corporate datacenters), the sampling rates remain rather high and thus the overall true positive and false positive counts remain mostly unaltered. On the other hand, when false negatives are relatively cheap (as is often the case for private, user owned desktops), we can expect our system to determine sampling rates that are relatively low.

D. Solution Times

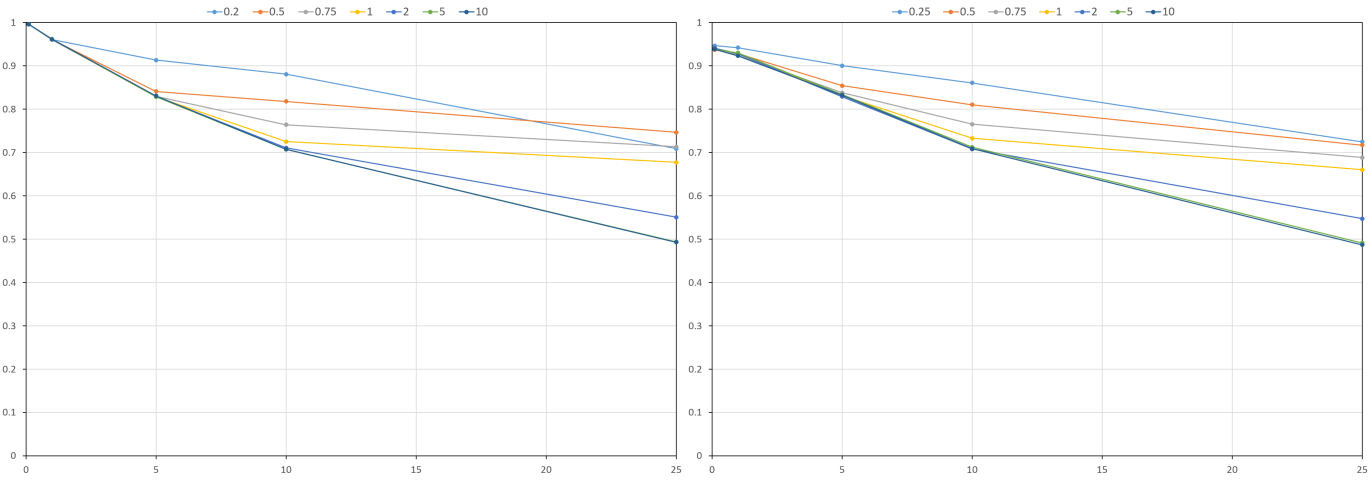
We recognize that for a system such as the one proposed in this paper to be applicable to real world scenarios, it is required that solving and computing the sampling rates be very fast and cheap. Long solving times mean that the system would not be able to quickly adapt to changing landscapes and to respond by setting new sampling rates in a timely fashion.

Figure 13 shows a cumulative distribution of the total solving times (both PuLP and Infer.NET) measured during



(a) without classifier overlap **(b)** with classifier overlap

Fig. 10: Percentage of true positives remaining compared to percentage of false positives eliminated. Different setting for FP-threshold and cost ratio correspond to different points on the curves.



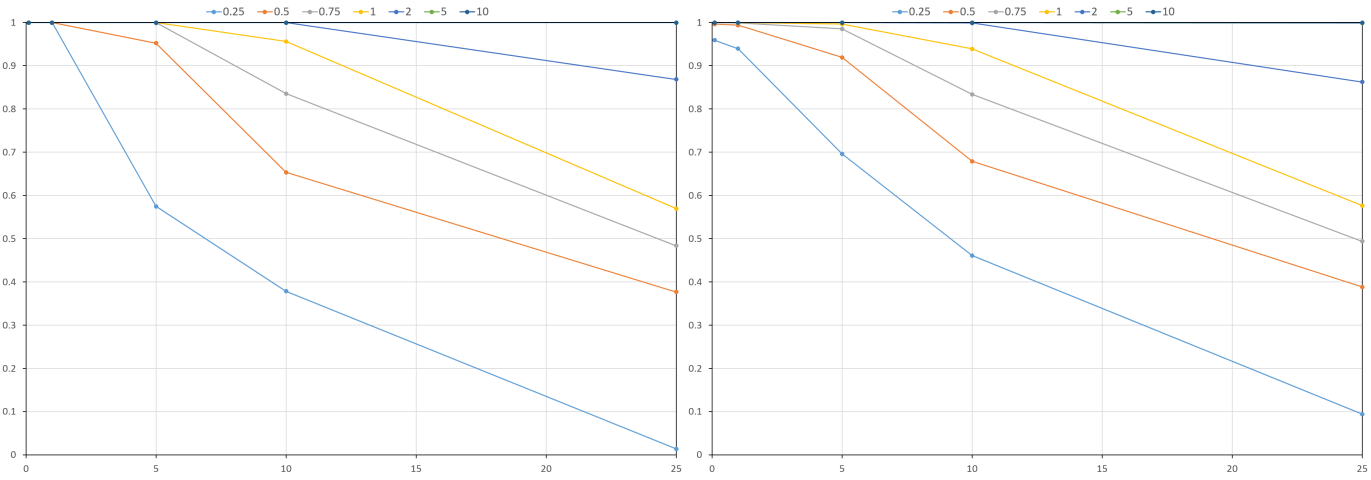
(a) without classifier overlap **(b)** with classifier overlap

Fig. 11: Classification precision as a function of θ for different values of β.

our simulations for each day. The figure shows that over 80% of simulated days were solved in under 20 seconds. The day which took our system the longest to solve took less than 5 minutes (285 seconds to be exact). This tells us that using this kind of system in a responsive manner is indeed feasible.

Figure 14 shows the average solving time needed for each day of our simulation. We first note that the solving times for PuLP (represented by the blue line) are extremely low, constantly below 1 second. When there is no classifier overlap, the solution provided by PuLP is sufficient as the sampling rates for the classifiers. This means that when there is no overlap, solving is extremely fast. Also, as can be seen from the figure, there is a clear correlation between Infer.NET solving time and the number of active signatures, which both follow the same trends. This correlation is interesting as it indicates that

- we can anticipate the solving time in advance, and
- we can accelerate the solving of days with a large number of active signatures using a "divide-and-conquer" approach, meaning we can split them to smaller batches and solve each one separately.



(a) without classifier overlap **(b)** with classifier overlap

Fig. 12: Classification recall as a function of θ for different values of β.

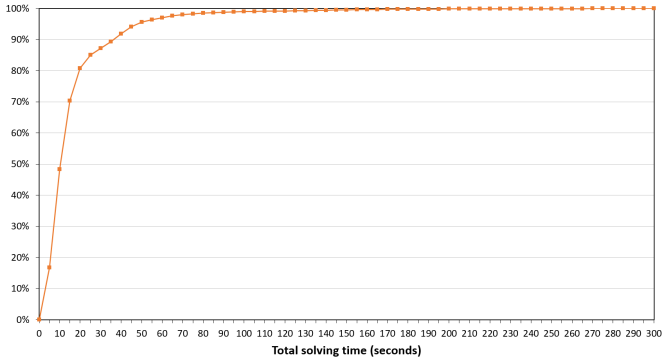


Fig. 13: Cumulative distribution of overall solution times (PuLP + Infer.NET).

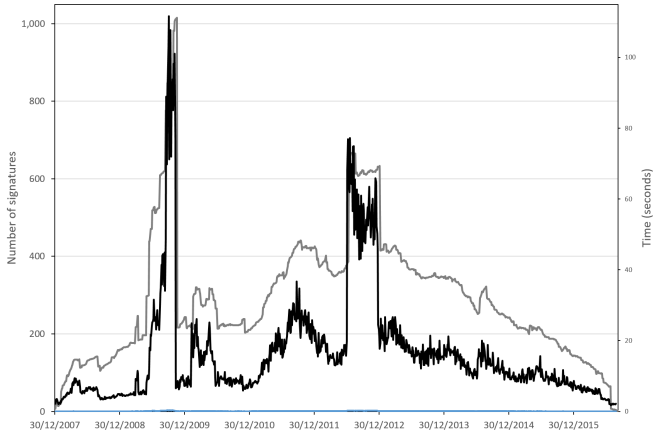


Fig. 14: Average daily solving times. The blue line (at the bottom) represents PuLP solving time ($\leq$ 1 second). The black line represent Infer.NET solving time. The grey line in the background show number of active signatures per day.

E. Experimental Summary

Reduction in false positives: Regardless of classifier overlap, when comparing the reduction in the number of false positives to that of true positives, we find that our responsive optimization technique removes *more* false positives, both in terms of percentages (19.23% compared to 12.39% without overlap; 20.13% compared to 11.96% with overlap) with overlap) and in terms of absolute values (9,286,530 compared to 8,002,871.5 without overlap; 9,225,422.6 compared to 8,065,888.6 with overlap). The reduction in absolute values is surprisingly significant, considering that the highest value of θ in our simulations was 25, meaning the false positive rate was initialized to 25% of the true positive rate. In settings with classifier overlap, applying sampling rates is *more beneficial* than in settings without classifier overlap. This can be observed from the relevant reduction rates, 20.13% compared to 19.23% of false positives and 11.96% compared to 12.39% of true positives. This means that, on average, we can eliminate more false positives at the expense of fewer true positives.

Solver running time: In settings *without* classifier overlap, the sampling rates for all simulated days were computed in mere seconds. In settings with overlap, timing measurements indicate that the proposed approach for setting sampling rates is computationally feasible and applicable to real-world settings. The measurements show that sampling rates for over 98% of simulated days were computed in under 2.5 minutes per day, with both daily average and median of 15 seconds.

V. Related Work

The closest work to ours focuses on reactive vs. proactive security in Section V-A, with subsequent sections discussing work on exploitation in the wild (Section V-B), models of malware propagation (Section V-C), and economics of security (Section V-D).

A. Reactive vs. Proactive Security

There has been some interest in comparing the relative performance of proactive and reactive security over the years. In the work closest to ours, Barth *et al.* [8] make the interesting observation that proactive security is not always more cost-effective than reactive security. Just like in our paper, they support their claim using simulations. Not sure if they take into account the long term effects of a short term investment in security (for example, implementing and integrating a new security mechanism is likely to be beneficial long after the investment in the mechanism has been reduced).

Barreto *et al.* study controllability and stability properties of dynamical systems when actuator or sensor signals are under attack [7]. They formulate a detailed adversary model that considers different levels of privilege for the attacker such as read and write access to information flows. They then study the impact of these attacks and propose reactive countermeasures based on game theory. In one case-study they use a basic differential game, and in the other case study they introduce a heuristic game for stability.

B. Exploitation In the Wild

Nayak *et al.* [39] highlights the lack of a clear connection between vulnerabilities and metrics such as the attack surface on the amount of exploitation that takes place. They focus on using field data to get a more comprehensive picture of the exploitation landscape as it is changing. They find that none of the products in their study have more than 35% of their disclosed vulnerabilities exploited in the wild. Furthermore, the exploitation ratio and the exercised attack surface tend to decrease with newer product releases. They also find that hosts that quickly upgrade to newer product versions tend to have reduced exercised attack-surfaces. These findings resonate with the premise of our paper.

Sabottke *et al.* focus on determining which vulnerabilities are likely to be exploited after a disclosure [48]. They conduct a quantitative and qualitative exploration of the vulnerability-related information disseminated on Twitter. They then describe the design of a Twitter-based exploit detector, and they introduce a threat model specific to our problem. In addition to response prioritization, their detection techniques have applications in risk modeling for cyber-insurance and they highlight the value of information provided by the victims of attacks.

Bilge *et al.* [12] focus on the prevalence and exploitation patterns of zero-days. They rely on identifying zero-day attacks from field-gathered data that records when benign and malicious binaries are downloaded on 11 million real hosts around the world. Searching this dataset for malicious files that exploit known vulnerabilities indicates which files appeared on the Internet before the corresponding vulnerabilities were disclosed. They identify 18 vulnerabilities exploited before disclosure, of which 11 were not previously known to have been employed in zero-day attacks. They also discover that a typical zero-day attack lasts 312 days on average and that, after vulnerabilities are disclosed publicly, the volume of attacks exploiting them increases by up to 5 orders of magnitude. Some of these findings were important in deciding on how to conduct credible simulations for our experimental evaluation in this paper.

Commercial Intelligence Reports: One of the better ways to understand the exploitation landscape is by consulting the intelligence reports that emerge from large security software vendors. Of these, reports published by Microsoft [37] and Symantec [54] stand out. Both are published on a regular basis, annually in the case of Microsoft and monthly in the case of Symantec.

A recent report from Microsoft presented at BlackHat highlights the importance of focusing on exploitation and not only vulnerabilities [59], [60]. The current approach to software security at Microsoft is driven by data. This approach involves proactive monitoring and analysis of exploits found in-the-wild to better understand the types of vulnerabilities that are being exploited and exploitation techniques being used. This category of analysis and insight has driven a series of mitigation improvements that has broken widely used exploitation techniques and in some cases virtually eliminated entire classes of vulnerabilities.

C. Models of Malware Propagation

Gao *et al.* attempt to model the propagation of mobile viruses [20]. They propose a two-layer network model for simulating virus propagation through both Bluetooth and SMS. Their work addresses the impacts of human behaviors, i.e., operational behavior and mobile behavior, on virus propagation. They provide some experimental results that show that their proposed strategies can effectively protect large-scale and highly dynamic mobile networks.

Bose *et al.* study four aspects crucial to modeling malware propagation: application-level interactions among users of such networks, local network structure, user mobility, and network coordination of malware such as botnets [13]. Since closed-form solutions of malware propagation considering these aspects are difficult to obtain, they describe an open-source, flexible agent-based emulation framework that can be used by malware researchers for studying today's complex malware. The framework, called Agent-Based Malware Modeling (AMM), allows different applications, network structure, network coordination, and user mobility in either a geographic or a logical domain to study various infection and propagation scenarios. In addition to traditional worms and viruses, the framework also allows modeling network coordination of malware such as botnets. The majority of the parameters used in the framework can be derived from real-life network traces collected from a network, and therefore, represent realistic malware propagation and infection scenarios. As representative examples, they examine two well-known malware spreading mechanisms: (i) A malicious virus, such as Cabir, spreading among the subscribers of a cellular network using Bluetooth, and (ii) A hybrid worm that exploit email and file-sharing to infect users of a social network. In both cases, they

identify the parameters most important to the spread of the epidemic based upon their extensive simulation results.

Fleizach *et al.* evaluate the effects of malware propagating using communication services in mobile phone networks [19]. Although self-propagating malware is well understood in the Internet, mobile phone networks have very different characteristics in terms of topologies, services, provisioning and capacity, devices, and communication patterns. The authors have developed an event-driven simulator that captures the characteristics and constraints of mobile phone networks. In particular, the simulator models realistic topologies and provisioned capacities of the network infrastructure, as well as the contact graphs determined by cell phone address books. Their evaluation focuses on the speed and severity of random contact worms in mobile phone networks, characterizing denial-of-service effects such worms could have on the network.

Garetto *et al.* present analytical techniques that can be used to better understand the behavior of malware, such as e-mail viruses and worms [21]. They develop a modeling methodology based on Interactive Markov Chains that is able to capture many aspects of the problem, especially the impact of the underlying topology on the spreading characteristics of malware. They propose numerical methods to obtain useful bounds and approximations in the case of very large systems, validating their results through simulation. An analytic methodology represents a fundamentally important step in the development of effective countermeasures for future malware activity.

Edwards *et al.* present a simple Markov model of malware spread through large populations of websites and studies the effect of two interventions that might be deployed by a search provider: blacklisting infected web pages by removing them from search results entirely and a generalization of blacklisting, called depreferencing, in which a website's ranking is decreased by a fixed percentage each time period the site remains infected [18]. They analyze and study the trade-offs between infection exposure and traffic loss due to false positives (the cost to a website that is incorrectly blacklisted) for different interventions. They find that interventions are most effective when websites are slow to remove infections. Surprisingly, they also find that low infection or recovery rates can increase traffic loss due to false positives. Their analysis also shows that heavy-tailed distributions of website popularity, as documented in many studies, leads to high sample variance of all measured outcomes. This result implies that it will be difficult to determine empirically whether certain website interventions are effective, and it suggests that theoretical models such as the one described in this paper have an important role to play in improving web security.

Grottke *et al.* define metrics and models for the assessment of coordinated massive malware campaigns targeting critical infrastructure sectors [25]. First, they develop an analytical model that allows us to capture the effect of neighborhood on different metrics (infection probability and contagion probability). Then, they assess the impact of putting operational but possibly infected nodes into quarantine. Finally, they study the implications of scanning nodes for early detection of malware (e.g., worms), accounting for false positives and false negatives. Evaluating this methodology using a small four-node topology, they find that malware infections can be effectively contained by using quarantine and appropriate rates of scanning for soft impacts.

Moore *et al.* study the abuse of "trending" search terms, in which miscreants place links to malware-distributing or ad-filled web sites in web search and Twitter results, by collecting and analyzing measurements over nine months from multiple sources [38]. They devise heuristics to identify ad-filled sites, report on the prevalence of malware and ad-filled sites in trending-term search results, and measure the success in blocking such content. They uncover collusion across offending domains using network analysis, and use regression analysis to conclude that both malware and ad-filled sites thrive on less popular, and less profitable trending terms. They also build an economic model informed by our measurements and conclude that ad-filled sites and malware distribution may be economic substitutes.

Cova *et al.* offer the first broad analysis of the infrastructure underpinning the distribution of rogue security software by tracking 6,500 malicious domains [14]. Secondly, they show how to apply attack attribution methodologies to correlate campaigns likely to be associated to the same individuals or groups. By using these techniques, they identify 127 rogue security software campaigns comprising 4,549 domains. Finally, they contextualize their findings by comparing them to a different threat ecosystem, that of browser exploits. They underline the profound difference in the structure of the two threats, and investigate the root causes of this difference by analyzing the economic balance of the rogue antivirus ecosystem. They track 372,096 victims over a period of 2 months and take advantage of this information to retrieve monetization insights. While applied to a specific threat type, the authors hypothesize that the methodology and the lessons learned from this work are of general applicability to develop a better understanding of the threat economies.

Hang *et al.* conduct an extensive study of malware distribution and follow a website-centric and user-centric point of view [26]. They collect data from four online databases, including Symantec's WINE Project, for more than 600K malicious URLs and over 500K users. They find that legitimate but compromised websites constitute 33.1% of the malicious websites in the dataset. In order to conduct this study, they develop a classifier to distinguish between compromised vs. malicious websites with an accuracy of 95.3%, which could be of interest to studies on website profiling. They find that malicious URLs can be surprisingly long-lived, with 10% of malicious sites staying active for over three months. Finally, the distribution of the visits to malicious sites per user is skewed, with 1.4% of users visiting more than 10 malicious sites in 8 months. Their study is a step toward modeling web-based malware propagation as a network-wide phenomenon and enabling researchers to develop realistic models.

Kwon *et al.* analyzed approximately 43,000 malware down-

load URLs [56]. Their measurement period is over 1.5 years in which they studied the URLs' long-term behavior. They discovered that some malware download sites survive for a very long time and revives many times, a fact that had not been revealed by previous research. They established three categories by focusing attention on malware variation. Their results showed that 10% of the unchanged category survives for more than 500 days and 10% of the changed occasionally category revives more than 15 times. They also analyzed sites in terms of change in IP address, number of anti-virus signatures, and URL features. They find that each category has different attacker operational and resource characteristics. Using these findings, they discuss how to mitigate the effects of each category

Arbaugh *et al.* [6] Introduced a vulnerability life-cycle model supported by case studies. The introduced model is different than the intuitive model one would imagine a vulnerability follows. We relied on the insights presented in this paper in designing our models for trace generation.

D. Economics of Security Attacks and Defenses

A report by the Ponemon Institute [43] estimated the costs of false positives to industry companies. The estimation was based on a survey filled by 18,750 people in various positions. While the numbers portrayed in the report are not accurate, they do paint an interesting picture. The average cost to false positives to a company was estimated at 1.27 million dollars per year. These estimations include the cost analyzing and investigating false positive reports as well the cost of not responding in time to other true positive reports.

Herley *et al.* [28] point out that, while it can be claimed that some security mechanism improves security, it is impossible to prove that a mechanism is necessary or sufficient for security, meaning there is no other way to prevent an attack or that no other mechanism is needed. They also make a similar observation stating that one can never prove that a security mechanism is redundant and not needed. These observations put into words the frame of mind that resulted in the current overwhelming number of active security mechanisms. We try to address this problem using the proposed sampling rates.

Online social networks (OSNs) offer a rich medium of malware propagation [29]. The authors monitor 3.5 million Facebook accounts and explore the role of pure monetary, social, and combined socio-monetary psychological incentives in OSN malware campaigns. The majority of the malware campaigns rely on pure social incentives. They also observe that malware campaigns using socio-monetary incentives infect more accounts and last longer than campaigns with pure monetary or social incentives. The latter suggests the efficiency of an epidemic tactic is surprisingly similar to the mechanism used by biological pathogens to cope with diverse gene pools.

Hardy *et al.* shed light on targeted malware attacks faced by organizations by studying malicious e-mails received by 10 civil society organizations (the majority of which are from groups related to China and Tibet issues) over a period of 4 years [27]. They find that the technical sophistication of malware they observe is fairly low, with more effort placed on socially engineering the e-mail content. They develop the Targeted Threat Index (TTI), a metric which incorporates both social engineering and technical sophistication when assessing the risk of malware threats. They demonstrate that this metric is more effective than simple technical sophistication for identifying malware threats with the highest potential to successfully compromise victims.

VI. Conclusions

In this paper we argued for a new kind of tunable framework on which to base security mechanisms. This new framework enables a more reactive approach to security allowing us to optimize the deployment of security mechanisms based on the current state of attacks. Based on actual evidence of exploitation collected from the field, our framework can choose which mechanisms to enable/disable so that we can minimize the overall costs and false-positive rates while maintaining a satisfactory level of security in the system.

Our responsive strategy is both computationally affordable and results in significant reductions in false positives, at the cost of introducing a moderate number of false negatives. Through measurements performed in the context of large-scale simulations we find that the time to find the optimal sampling strategy is mere seconds for the non-overlap case and under 2.5 minutes in 98% of overlap cases. The reduction in the number of false positives is significant (about 9.2 million removed over traces that are about 9 years long). The reduction in false positive rates is 20.13% and 19.23% with and without overlap, respectively.

Acknowledgments

We want to thank Matt Miller and members of the IE team at Microsoft, Brian Witten and Leyla Bilge at Symantec, Amichai Shulman at Imperva.com, Jeremiah Grossman at SentinelOne, Tudor Dumitraş at UMD, Hovav Shacham at UCSD, as well as a number of others for useful discussions related to this work that ultimately allowed us to refine our understanding of the attack space.

References

[1] ClamAV. http://www.clamav.net/.

[2] Infer.NET. http://research.microsoft.com/en-us/um/cambridge/projects/infernet/.

[3] PuLP. https://pypi.python.org/pypi/PuLP.

[4] Separation of mechanism and policy. https://en.wikipedia.org/wiki/Separation_of_mechanism_and_policy, 2016.

[5] L. Allodi and F. Massacci. Comparing vulnerability severity and exploits using case-control studies. *In ACM Transactions on Information and System Security*, 2014.

[6] W. A. Arbaugh, W. L. Fithen, and J. McHugh. Windows of vulnerability: A case study analysis. *In Computer*, 2000.

[7] C. Barreto, A. A. Cárdenas, and N. Quijano. Controllability of dynamical systems: Threat models and reactive security. In *In Proceedings of the International Conference on Decision and Game Theory for Security*, 2013.

[8] A. Barth, B. I. P. Rubinstein, M. Sundararajan, J. C. Mitchell, D. Song, and P. L. Bartlett. A learning-based approach to reactive security. In *In Proceedings of the International Conference on Financial Cryptography and Data Security*, 2010.

[9] D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in client-side xss filters. In *In Proceedings of the International Conference on World Wide Web*, 2010.

[10] A. Baumhof. Banking malware at its best: a detailed llook at a new zeus/wsnpoem (zbot) variant. http://www.tidos-group.com/blog/2009/01/20/banking-malware-at-its-best-a-detailed-look-at-a-new-zeuswsnpoem-zbot-variant/, 2009.

[11] L. Bilge and T. Dumitraș. Investigating zero-day attacks. *In ;login:*, 2013.

[12] L. Bilge and T. Dumitras. Before we knew it: An empirical study of zero-day attacks in the real world. In *In Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[13] A. Bose and K. G. Shin. Agent-based modeling of malware dynamics in heterogeneous environments. *In Security and Communication Networks*, 2013.

[14] M. Cova, C. Leita, O. Thonnard, A. D. Keromytis, and M. Dacier. *An Analysis of Rogue AV Campaigns*. 2010.

[15] T. H. Dang, P. Maniatis, and D. Wagner. The performance cost of shadow stacks and stack canaries. In *In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[16] G. De Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna. Pexy: The other side of exploit kits. In *In Proceedings of the SIG SIDAR conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2014.

[17] N. K. Dien, T. T. Hieu, and T. N. Thinh.

[18] B. Edwards, T. Moore, G. Stelle, S. Hofmeyr, and S. Forrest. Beyond the blacklist: modeling malware spread and the effect of interventions. In *In Proceedings of the Workshop on New Security Paradigms*, 2012.

[19] C. Fleizach, M. Liljenstam, P. Johansson, G. M. Voelker, and A. Mehes. Can you infect me now?: malware propagation in mobile phone networks. In *In Proceedings of the ACM Workshop on Recurring malcode*, 2007.

[20] C. Gao and J. Liu. Modeling and restraining mobile virus propagation. *In IEEE Transactions on Mobile Computing*, 2013.

[21] M. Garetto, W. Gong, and D. Towsley. Modeling malware spreading dynamics. In *In Proceedings of the IEEE International Conference on Computer Communications*, 2003.

[22] I. Gashi, V. Stankovic, C. Leita, and O. Thonnard. An experimental study of diversity with off-the-shelf antivirus engines. In *In Proceedings of the IEEE International Symposium on Network Computing and Applications*, 2009.

[23] GReAT. Projectsauron: top level cyber-espionage platform covertly extracts encrypted government comms. https://securelist.com/analysis/publications/75533/faq-the-projectsauron-apt/, 2016.

[24] K. Griffin, S. Schneider, X. Hu, and T.-C. Chiueh. Automatic generation of string signatures for malware detection. In *In Proceedings of the international conference on Recent Advances in Intrusion Detection*, 2009.

[25] M. Grottke, A. Avritzer, D. S. Menasché, J. Alonso, L. Aguiar, and S. G. Alvarez. Wap: Models and metrics for the assessment of critical-infrastructure-targeted malware campaigns. In *In Proceedings of the IEEE International Symposium on Software Reliability Engineering*, 2015.

[26] H. Hang, A. Bashir, M. Faloutsos, C. Faloutsos, and T. Dumitras. "infect-me-not": A user-centric and site-centric study of web-based malware. In *In Proceedings of the IFIP Networking Conference and Workshops*, 2016.

[27] S. Hardy, M. Crete-Nishihata, K. Kleemola, A. Senft, B. Sonne, G. Wiseman, P. Gill, and R. J. Deibert. Targeted threat index: Characterizing and quantifying politically-motivated targeted malware. In *In Proceedings of the USENIX Security Symposium*, 2014.

[28] C. Herley. Unfalsifiability of security claims. *In Proceedings of the National Academy of Sciences*, 2016.

[29] T.-K. Huang, B. Ribeiro, H. V. Madhyastha, and M. Faloutsos. The socio-monetary incentives of online social network malware campaigns. In *In Proceedings of the ACM Conference on Online Social Networks*, 2014.

[30] V. Ivanov. Protocol-level evasion of web application firewalls, 2012.

[31] T. Jarrett. The fog of war: How prevalent is sql injection? https://www.veracode.com/blog/2015/07/fog-war-how-prevalent-sql-injection, 2015.

[32] C. Kerschbaumer. Enforcing content security by default within Web browsers. In *In Proceedings of the IEEE Cybersecurity Development Conference*, 2015.

[33] V. Kotov and F. Massacci. Anatomy of exploit kits: Preliminary analysis of exploit kits as software artefacts. In *In Proceedings of the International Conference on Engineering Secure Software and Systems*, 2013.

[34] L. M. L. Radvilavicius and A. Cenys. Overview of real-time antivirus scanning engines. *In Journal of Engineering Science and Technology Review*, 2012.

[35] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf. Policy/mechanism separation in hydra. *In SIGOPS Operating Systems Review*, 1975.

[36] H. A. Loeliger. An introduction to factor graphs. *In IEEE Signal Processing Magazine*, 2004.

[37] Microsoft Corporation. Microsoft intelligence report. https://www.microsoft.com/security/sir/default.aspx, 2015.

[38] T. Moore, N. Leontiadis, and N. Christin. Fashion crimes: Trending-term exploitation on the web. In *InProceedings of the ACM Conference on Computer and Communications Security*, 2011.

[39] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitraş. *Some Vulnerabilities Are Different Than Others*. 2014.

[40] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *In Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[41] M. Payer. Too much pie is bad for performance. 2012.

[42] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *In Proceedings of the Netwroked Systems Design and Implementation*, 2010.

[43] Ponemon Institute. The cost of malware containment. http://www.ponemon.org/local/upload/file/Damballa%20Malware%20Containment%20FINAL%203.pdf, 2015.

[44] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iFRAMEs point to us. In *In Proceedings of the Usenix Security Symposium*, 2008.

[45] N. Provos, M. A. Rajab, and P. Mavrommatis. Cybercrime 2.0: when the cloud turns dark. *In Communications of the ACM*, 2009.

[46] P. Roberts. At the vulnerability Oscars, the winner is... buffer overflow!! https://www.veracode.com/blog/2013/02/at-the-vulnerability-oscars-the-winner-is-buffer-overflow, 2013.

[47] N. J. Rubenking. False positives sink antivirus ratings. http://www.pcmag.com/article2/0,2817,2481367,00.asp, 2015.

[48] C. Sabottke, O. Suciu, and T. Dumitraș. Vulnerability disclosure in the age of social media: exploiting twitter for predicting real-world exploits. In *In USENIX Security Symposium*, 2015.

[49] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar. Signature generation and detection of malware families. In *In Australasian Conference on Information Security and Privacy*, 2008.

[50] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. Machine learning: The high interest credit card of technical debt. In *In SE4ML: Software Engineering for Machine Learning (Neural Information Processing Systems Workshop)*, 2014.

[51] B. Stock, B. Livshits, and B. Zorn. Kizzle: A signature compiler for exploit kits. In *In Proceedings of the International Conference on Dependable Systems and Networks*, 2016.

[52] O. Sukwong, H. S. Kim, and J. C. Hoe. Commercial antivirus software effectiveness: an empirical study. *In Computer*, 2011.

[53] Symantec Corporation. Insight: Deployment best practices. https://support.symantec.com/en_US/article.DOC5077.html, 2016.

[54] Symantec Corproration. Internet security threat report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf, 2016.

[55] L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *In Proceedings of the IEEE Symposium on Security and Privacy*, 2013.

[56] Y. Tanaka and A. Goto. Analysis of malware download sites by focusing on time series variation of malware. In *In Proceedings of the IEEE Symposium on Computers and Communication (ISCC)*, 2016.

[57] Tenable. Tenable malware detection: Keeping up with an increasingly sophisticated threat environment. http://www.enpointe.com/images/pdf/whitepaper-tenable-malware-detection.pdf, 2014.

[58] D. Weston and M. Miller. Web application firewalls: Attacking detection logic mechanisms, 2016.

[59] D. Weston and M. Miller. Windows 10 mitigation improvements, 2016.

[60] D. Weston, M. Miller, and T. Rains. Exploitation trends: From potential risk to actual risk. https://www.rsaconference.com/writable/presentations/file_upload/br-t07-exploitation-trends-from-potential-risk-to-actual-risk.pdf, 2015.

[61] M. F. Zolkipli and A. Jantan. A framework for malware detection using combination technique and signature generation. In *In Computer Research and Development*, 2010.