Data Driven Game Theoretic Cyber Threat Mitigation

John Robertson, Vivin Paliath, Jana Shakarian, Amanda Thart, Paulo Shakarian[*]
Arizona State University

Abstract

Penetration testing is regarded as the gold-standard for understanding how well an organization can withstand sophisticated cyber-attacks. However, the recent prevalence of markets specializing in zero-day exploits on the darknet make exploits widely available to potential attackers. The cost associated with these sophisticated kits generally precludes penetration testers from simply obtaining such exploits – so an alternative approach is needed to understand what exploits an attacker will most likely purchase and how to defend against them. In this paper, we introduce a data-driven security game framework to model an attacker and provide policy recommendations to the defender. In addition to providing a formal framework and algorithms to develop strategies, we present experimental results from applying our framework, for various system configurations, on real-world exploit market data actively mined from the darknet.

1 Introduction

Many corporations rely on extensive penetration testing to assess the security of their computer networks. In a penetration test, a red team is hired to expose major flaws in the firms security infrastructure. Recently, however, the market for exploit kits has continued to evolve and what was once a rather hard-to-penetrate and exclusive market – whose buyers were primarily western governments (Shakarian, Shakarian, and Ruef 2013), has now become more accessible to a much wider population. Specifically, the darknet portions of the Internet accessible through anonymization protocols such as Tor and i2p – have become populated with a variety of markets specializing in such products (Shakarian and Shakarian 2015; Ablon, Libicki, and Golay 2014). In particular, 2015 saw the introduction of darknet markets specializing in zero-day exploit kits – exploits designed to leverage previously undiscovered vulnerabilities. These exploit kits are difficult and time-consuming to develop – and often are sold at premium prices. We have surveyed 8 marketplaces and show the price ranges of exploit kits for common software in Table 1 – these range from 0.0126-8.4 Bitcoin (2.88-1919.06 U.S. dollars at the time of this writing).

[*]shak@asu.edu

The widespread availability of zero-day exploits represents a potential game changer for penetration testers – specifically posing the following questions:

- *What exploits will an attacker likely purchase if he targets my organization?*
- *What software used in the organization pose the biggest risk to new threats?*

However, the high cost of a variety of exploits available on the darknet may preclude a penetration tester from simply obtaining them. In this paper, we introduce a novel, data-driven security game framework to directly address these challenging questions. Given a system configuration (or a distribution of system configurations within an organization) we model an attacker who, given a budget, will purchase exploits to maximize his level of access to the target system. Likewise, a defender will look to adjust system configurations in an effort to minimize the effectiveness of an attacker while ensuring that necessary software dependencies are satisfied. Not only have we introduced a rigorous and thoroughly analyzed framework for these problems, but we have also implemented and evaluated a system that is fed with real-world exploit market data, mined from the darknet. We are currently moving our system toward real-time scraping of market information to provide game-theoretic assessment of the exploit market – while considering specific system information. We provide a schematic diagram of our system in Figure 1. The specific contributions of this paper include a new security game framework designed to model an attacker with access to exploit markets and a defender of information technology infrastructure (Section 2); theoretical analysis of the framework leading to the development of algorithms to find near-optimal strategies for both players (Section 3); and an implementation of the system and the results of a thorough suite of experiments on real-world data (Section 4). Before discussing these contributions, we review some domain-specific background and related literature in the security games.

Exploit markets on the darknet. While criminal activity on the darknet has been extensively studied over the past decade for issues such as drug trade (Soska and Christin 2015) and terrorism (Chen 2011) the markets of exploits existing on the darknet are much less well-understood. There has been related work on malicious hacker forums (Zhao et al. 2012;

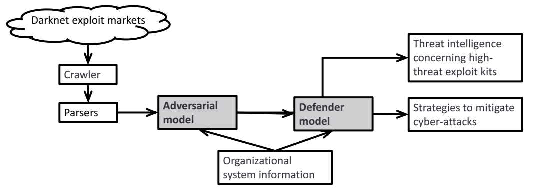
Darknet exploit markets
Crawler
Parsers
Adversarial model
Defender model
Threat intelligence concerning high-threat exploit kits
Strategies to mitigate cyber-attacks
Organizational system information

Figure 1: Schematic of real-time exploit analysis system.

Product	**Price in BTC**	**Price in $***
GovRAT (Source Code + 1 Code Signing Certificate Included)	2.000	$456.92
0day Wordpress MU Remote Shell	1.500	$342.69
A5/1 Encryption Rainbow Tables	1.500	$342.69
Unlimited Code Signing Certificate	1.200	$274.16
Ready-made Linux botnet 600 SERVERS	1.200	$274.16
FUD version of Adobe Flash <=16.0.0.287 (CVE 2015-0311)	1414.68	$600.00

*Price in U.S. Dollar as of Sep. 1, 2015 [1 BTC = $228.46]

Table 1: Example of Products offered on Darknet Markets

Li and Chen 2014), which did not focus on the purchase and sale of specific items. Markets of malicious products relevant to cyber security have been previously studied (Ablon, Libicki, and Golay 2014; Shakarian and Shakarian 2015), but none of these works gathered data on specific exploits (or other products) from either the darkweb or open Internet; nor did they examine the markets through the lens of security games. To our knowledge, this is the first work that describes the collection of price data on specific exploits for sale on the deep web and then analyzes them in a security game framework that yields policy recommendations for cyber defenders that are tailored to specific system configurations.

Related work in security games. In recent years, "security games" where attacker-defender models are used to inform the actions of defenders in military, law-enforcement, and homeland security applications have gained much traction (see (Tambe 2011) for an overview). With regard to cyber-security, there have been many contributions including intrusion detection (Nguyen, Alpcan, and Başar 2009); attack graph based games (Lye and Wing 2005) and honeypot placement (Kiekintveld, Lisý, and Píbil 2015). However, to the best of our knowledge, the work of this paper represents the first game theoretic approach to *host-based* defense where the activities of the attacker are informed from an "unconventional" source (information not directly related to the defender's system) - specifically information from darknet markets in this case. Further, the very recent emergence of darknet markets specializing in zero-day exploits allow for the integration of information that was unavailable in previous work.

2 Security Game Framework

Here we formalize our concept of our security game where the attacker is a malicious hacker with access to a darknet exploit market and the defender is tasked with host-based defense of either a single or group of systems. We use the notation V to represent the entire set of vulnerabilities within a given computer system. Though there may be vulnerabilities not yet detected by the system administrator, we can mine for information on new vulnerabilities through an examination of darknet hacking markets. In a real-world organization, system administrators are not able to patch all vulnerabilities for a variety of reasons. Software dependencies, use of legacy systems, and non-availability of patches are some examples. To model this, we define a "constraint set" (denoted C) as a subset of V. The vulnerabilities in a constraint set represent the vulnerabilities required for some system functionality. When each vulnerability in a constraint set C is in the presented attack surface (i.e. externally accessible), C is then said to be satisfied and the system supports the functionality modeled by C. Let $\mathbf{C}$ represent the set of all possible constraint sets. We extend this idea with an "application constraint set" which, for an arbitrary application, i, denoted $\mathcal{C}_i$, is a set of constraint sets (i.e. $\mathcal{C}_i \subseteq \mathbf{C}$). Each constraint set in $\mathcal{C}_i$ represents a set of vulnerabilities that together will provide complete functionality of application i. $\mathcal{C}_i$ is said to be satisfied if any single constraint set in $\mathcal{C}_i$ is satisfied. If $\mathcal{C}_i$ is satisfied by a system configuration, and hence at least one constraint set in $\mathcal{C}_i$ is satisfied, application i will properly operate on the system. $\mathcal{C}$ is the set of all application constraint sets for a given system configuration and represents all of the applications to be run on the system. So, in this framework, for a given system, a system administrator must select which vulnerabilities must be present in order to allow each application i to function. This begs the question as to how to make this selection – so we now start to define some concepts relevant to the adversary.

We will use ex to denote a particular exploit - a technique used to take advantage of a given vulnerability. Let Ex denote the set of all possible exploits and $\mathbf{Ex}$ denote the set of all possible exploit sets (i.e. $\mathbf{Ex} = 2^{Ex}$). For each $ex \in Ex$, c_{ex} is the associated cost of exploit ex - and this is specified directly on a darknet market (normally in Bitcoin). Associated with the set of exploits is the Exploit Function, ExF, which takes a set of exploits as input and returns a set of vulnerabilities (i.e. $ExF : \mathbf{Ex} \to 2^V$). The set of vulnerabilities produced by $ExF(A)$, for a given set of exploits A, represents the vulnerabilities that are exploited by the exploits in A. While many possible variations of an exploit function are possible, in this paper, we will use a straightforward definition that extends the exploit function from singletons (whose associated vulnerabilities can be taken directly from the online marketplaces) to sets of exploits: $ExF(A) = \bigcup_{a \in A} ExF(\{a\})$. For use in proving complexity results, we shall denote the special case where $Ex = V$, $ExF(A) = A$, and $\forall ex \in Ex,\ c_{ex} = 1$ as the "Identity Exploit Model".

Player Strategies and Payoff. An attacker will use a set of exploits to attempt to gain access to a system, and must do so within a budget. Likewise, the defender must identify a set of vulnerabilities that he is willing to expose (often referred to as the "presented attack surface"). We define strategies for the two players formally as follows.

Definition 2.1. (Attack Strategy). Given budget $k_{atk} \in \mathbb{R}^+$, an *Attack Strategy*, denoted A is a subset of Ex s.t. $\sum_{a \in A} c_a \le k_{atk}$.

Definition 2.2. (Defense Strategy). Given a family of application constraint sets $\mathcal{C} = \{\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n\}$, a *Defense Strategy*, denoted D is a subset of V s.t. for each $\mathcal{C}_i \in \mathcal{C}$, there exists $C \in \mathcal{C}_i$ where $C \subseteq D$.

Note that when a defense strategy D meets the requirements of $\mathcal{C}$, as per Definition 2.2, we say D *satisfies* $\mathcal{C}$. We will use the notation $\mathbf{A}, \mathbf{D}$ to denote the set of all attack and defense strategies, respectively, and refer to an attacker-defender pair of strategies as a "strategy profile." We will also define a *mixed strategy* for both players in the normal manner. For the attacker (resp. defender) a mixed strategy is a probability distribution over $\mathbf{A}$ (resp. $\mathbf{D}$). We shall normally denote mixed strategies as Pr_A, Pr_D for each player and use the notation $|Pr_A|$ (resp. $|Pr_D|$) to denote the number of strategies in $\mathbf{A}$ (resp. $\mathbf{D}$) that are assigned a nonzero probability by the mixed strategy. We now turn our attention to the payoff function, which we define formally as follows:

Definition 2.3. (Payoff Function). A payoff function, p, is any function that takes a strategy profile as an argument and returns a positive real. Formally, $p : \mathbf{A} \times \mathbf{D} \to \mathbb{R}^+$

Unless noted otherwise, we will treat the payoff function as being computable in polynomial time. Also, the payoff function is underspecified - which is designed to allow flexibility in the framework. However, in the context of the results of this paper, we shall consider the following "payoff function axioms":

$$\forall D \in \mathbf{D}, \forall A \in \mathbf{A} \text{ s.t. } ExF(A) \cap D = \emptyset,\ p(A, D) = 0 \quad (1)$$

$$\forall D \in \mathbf{D},\ \forall D' \subseteq D\ , \forall A \in \mathbf{A},\ p(A, D') \le p(A, D) \quad (2)$$

$$\forall D \in \mathbf{D},\ \forall A \in \mathbf{A},\ \forall A' \subseteq A,\ p(A', D) \le p(A, D) \quad (3)$$

$$\forall A \in \mathbf{A},\ D, D' \in \mathbf{D}, p(A, D) + p(A, D') \ge p(A, D \cup D') \quad (4)$$

$$\forall D \in \mathbf{D},\ A, A' \in \mathbf{A}, p(A, D) + p(A', D) \ge p(A \cup A', D) \quad (5)$$

Axiom 1 states that if the vulnerabilities generated by an attack strategy's exploits and the vulnerabilities in a defense strategy are disjoint sets, the payoff function must return 0. A consequence of axiom 1 is that if either the attack strategy or the defense strategy is the empty set, the payoff function will return 0. Axioms 2 and 3 require the payoff function to be monotonic in the size of the attack and defense strategies. Axioms 4 and 5 require the payoff function to be submodular with respect to the attack and defense strategies.

In this paper, we shall (in general) focus on the "overlap payoff function" which we shall define as follows: $p(A, D) = |ExF(A) \cap D|$. Intuitively, this is simply the number of vulnerabilities exploited by the attacker. Further, when dealing with mixed strategies, we shall discuss payoff in terms of expectation. Expected payoff can be formally defined as follows:

$$Exp(Pr_A, Pr_D) = \sum_{D \in \mathbf{D}} \sum_{A \in \mathbf{A}} Pr_A(A) Pr_D(D) p(A, D)$$

Using the overlap function, the expected payoff can be interpreted as the "expected number of exploited vulnerabilities."

Best Response Problems. We now have the components to define a pair of decision problems dealing with the best response for the players. These problems are the deterministic host attacker problem (DHAP) and deterministic host defender problem (DHDP), respectively, and are defined as follows:

DHAP.
INPUT: $k_{atk} \in \mathbb{R}^+$, $x \in \mathbb{R}^+$, mixed defense strategy Pr_D, and payoff function p.
OUTPUT: "Yes" if $\exists A \in \mathbf{A}$ s.t. $\sum_{a \in A} c_a \le k_{atk}$, and $\sum_{D \in \mathbf{D}} Pr_D(D) p(A, D) \ge x$, "No" otherwise.

DHDP.
INPUT: $x \in \mathbb{R}^+$, application constraints $\mathcal{C}$, mixed attack strategy Pr_A, and payoff function p.
OUTPUT: "Yes" if $\exists D \in \mathbf{D}$ s.t. $\sum_{A \in \mathbf{A}} Pr_A(A) p(A, D) \le x$ and D satisfies $\mathcal{C}$. and "No" otherwise.

The natural optimization variants for these two problems will deal with maximizing the payoff in DHAP and minimizing the payoff in DHDP.

3 Analysis and Algorithms

In this section we analyze the complexity and limits of approximation for both DHAP and DHDP. We use the "Identity Exploit Model" for the complexity results. Unfortunately, both problems are NP-Complete in the general case.

Theorem 1. DHAP is NP-Complete, even when $|Pr_D| = 1$ and the payoff function adheres to the submodularity and monotonicity axioms.

Proof Sketch. Membership in NP is trivial if the payoff is PTIME computable. The hardness result relies on an embedding of the well-known budgeted set cover (Feige 1998). Here, the defender's strategy is treated as a set of elements to cover and the exploits are treated as subsets of D (by virtue of the exploit function). Exploit costs are set as 1 and the attacker's budget is the value budget from the embedded problem. So, the attacker must pick exploits to meet the budget and cover the determined number of the defender's vulnerabilities.■

Theorem 2. When $|\mathcal{C}| > 1$ and $|Pr_A| = 1$, DHDP is NP-Complete.

Proof Sketch. Again, membership in NP is trivial if the payoff is PTIME computable. Hardness is shown by embedding the hitting set problem. In this reduction, the attacker plays all exploits and each exploit corresponds with precisely one vulnerability. This has the effect of imposing a unit cost on each vulnerability. Here, each $\mathcal{C}_i$ must be covered by a vulnerability. Hence, the defender must pick a set of all vulnerabilities to meet the cost requirement of DHDP while covering each $\mathcal{C}_i$.■

We also were able to analyze the hardness of approximation for the optimization variants of DHAP and DHDP. Due to the fact that the above embeddings used set cover and hitting set, we can draw upon the results of (Feige 1998) to obtain the following corollaries:

Corollary 3. DHAP can not be approximated where the payoff is within a factor of $(1 - \frac{1}{e})$ unless $P = NP$

Corollary 4. DHDP can not be approximated where the payoff is within a factor of $(1-o(1))ln(n)$ unless $P = NP$

With the limits of approximation in mind, we can now introduce several algorithms to solve the optimization variants of DHAP and DHDP. The optimization variant of DHAP under the overlap payoff function is a special case of submodular maximization with the distinction that we are not simply picking k discrete objects, but instead picking items that each have a unique cost associated with them. Understanding this, we examine several different approaches to this problem based on the literature on submodular maximization. DHDP, on the other hand, can be readily approximated using the traditional set-cover algorithm (under some realistic assumptions), as cost does not affect DHDP.

Algorithm 1 Lazy Greedy Algorithm (Cost-Benefit Variant)

```
Input: $k_{atk} \in \mathbb{R}^+$, $Pr_D$, and payoff function $p$.
Output: $A \subseteq Ex$ s.t. $\sum_{a \in A} c_a \leq k_{atk}$
 1: $A \leftarrow \emptyset$; $cost \leftarrow 0$; priority queue $Q \leftarrow \emptyset$; $iter \leftarrow 1$
 2: for $e \in Ex$ do
 3:     $e.key \leftarrow \frac{\Delta_{p,Pr_D}(e|\emptyset)}{c_e}$; $e.i \leftarrow 1$
 4:     insert $e$ into $Q$ with "$key$" as the key
 5: end for
 6: while $\{a \in Ex \backslash A : c_a + cost \leq k_{atk}\} \neq \emptyset$ do
 7:     extract top (max) element $e$ of $Q$
 8:     if $e.i = iter$ and $c_e$ + cost $\leq k_{atk}$ then
 9:         $A \leftarrow A \cup \{e\}$; $iter \leftarrow iter + 1$
10:         $cost \leftarrow cost + c_e$
11:     else if $c_e$ + cost $\leq k_{atk}$ then
12:         $e.i \leftarrow iter$; $e.key \leftarrow \frac{\Delta_{p,Pr_D}(e|A)}{c_e}$;
13:         re-insert $e$ into $Q$
14:     end if
15: end while
16: return $A$
```

Algorithms for DHAP.
Greedy Approaches. As mentioned earlier, the non-unit cost of exploits mean that DHAP can be considered as a submodular maximization problem subject to knapsack constraints. Two versions of the traditional greedy algorithm (Nemhauser, Wolsey, and Fisher 1978) can be applied: a cost-benefit variant and uniform-cost variant, both of which will also use the lazy-greedy optimization (Minoux 1978) to further enhance performance while maintaining the approximation guarantee. We note that independently, the uniform-cost and the cost-benefit algorithms can perform arbitrarily badly. However, by extending a result from (Leskovec et al. 2007), either the cost-benefit or the uniform-cost algorithm will provide a solution within a factor of $\frac{1}{2}(1-1/e)$ for a given set of input parameters. By applying both algorithms to a given problem instance and returning the attack strategy which produces the larger payoff, the $\frac{1}{2}(1-1/e)$ approximation factor is achieved for DHAP. A cost-benefit lazy approximation algorithm is shown in Algorithm 1. By removing "c_e" from the denominator in the $e.key$ assignment in lines 3 and 12, the cost benefit lazy approximation algorithm is transformed into a uniform cost lazy approximation algorithm.
Multiplicative Update Approach. An improved approximation ratio, when compared with the $\frac{1}{2}(1-1/e)$ ratio for the greedy algorithms, can be obtained by adapting Algorithm 1 from (Azar and Gamzu 2012) for DHAP. This is shown as Algorithm 2 in this paper. For some value ϵ (a parameter), this algorithm provides a $(1-\epsilon)(1-1/e)$ approximation of the optimal solution (Theorem 1.2 in (Azar and Gamzu 2012)), which, by providing an exceedingly small ϵ value, can get arbitrarily close to the $(1-1/e)$ optimal approximation limit we discussed earlier.

Algorithm 2 Multiplicative Update

```
Input: $k_{atk}, \epsilon \in \mathbb{R}^+$ s.t. $0 < \epsilon \leq 1$, $Pr_D$, and payoff function $p$.
Output: $A \subseteq Ex$ s.t. $\sum_{a \in A} c_a \leq k_{atk}$
 1: $Ex' \leftarrow \{ex \in Ex : c_{ex} \leq k_{atk}\}$
 2: $A \leftarrow \emptyset$
 3: $W \leftarrow \min_{ex'_i \in |Ex'|} k_{atk}^2 / c_{ex'_i}$
 4: $w \leftarrow 1/k_{atk}$; $\lambda \leftarrow e^{\epsilon W/4}$
 5: while $k_{atk} w \leq \lambda$ and $Ex' \neq \emptyset$ do
 6:     $ex_j \leftarrow \text{argmin}_{ex_j \in Ex' \backslash A} \frac{c_{ex_j}}{k_{atk}} w / \Delta_{p,Pr_D}(ex_j|A)$
 7:     $A \leftarrow A \cup \{ex_j\}$
 8:     $w \leftarrow w\lambda^{c_{ex_j}/k_{atk}^2}$
 9:     $Ex' \leftarrow Ex' \backslash \{ex_j\}$
10: end while
11: if $\sum_{A_i \in A} c_{A_i} \leq k_{atk}$ then
12:     return $A$
13: else if $\sum_{D \in Pr_D} Pr_D(D) p(A \backslash \{ex_j\}, D) \geq$
        $\sum_{D \in Pr_D} Pr_D(D) p(\{ex_j\}, D)$ then
14:     return $A \backslash \{ex_j\}$
15: else
16:     return $\{ex_j\}$
17: end if
```

Algorithm for DHDP. When using the overlap payoff function, DHDP can be modeled as a weighted set cover problem. Because the overlap payoff function is a modular function, the associated cost of a given vulnerability v, is simply the payoff produced by the singleton set $\{v\}$ with a mixed attack strategy Pr_A (i.e. $c_v = \sum_{A \in Pr_A} Pr_A(A) p(A, \{v\})$. In the common case where each constraint set is a singleton set (i.e. $\forall \mathcal{C}_i \in \mathcal{C}$, $\forall C \in \mathcal{C}_i$, $|C| = 1$), if the overlap payoff function is used, an adaptation on the standard greedy weighted set cover algorithm can be used for DHDP (Algorithm 3), providing a $ln(n)+1$ approximation (Feige 1998).

4 Evaluation and Discussion

Darknet Market Data. We scraped and parsed eight marketplaces located on the Tor network during the month of May 2015. Each of these markets host vendors offering "hacking tools", including malware, botnets, exploits and other means serving to breach, steal and otherwise manipulate virtual targets. The product list is comprised of 235 such hacking tools, 167 of which were distinct. We found several

Algorithm 3 Weighted Greedy DHDP Algorithm for Singleton Constraint Set and Overlap Payoff Function Case

```
Input: Vulnerabilities $V$, $Pr_A$, and application constraints $\mathcal{C}$.
Output: $D \subseteq V$ s.t. the application constraints $\mathcal{C}$ are satisfied.
 1: $D \leftarrow \emptyset$
 2: $S \leftarrow$ set s.t. $S_i = \{j : V_i \in \mathcal{C}_j$ where $V_i$ is $i$th vulnerability in $V$ $\}$
 3: $c_{S_i} \leftarrow \sum_{A \in Pr_A} Pr_A(A)|ExF(A) \cap \{V_i\}|$
 4: $\mathcal{C}' \leftarrow [|\mathcal{C}|]$
 5: while $\mathcal{C}' \neq \emptyset$ do
 6:     $S_i \leftarrow \text{argmax}_{S_i \in S} \frac{|S_i \cap \mathcal{C}'|}{c_{S_i}}$
 7:     $\mathcal{C}' \leftarrow \mathcal{C}' \backslash S_i$
 8:     $D \leftarrow D \cup \{V_i\}$
 9: end while
10: return $D$
```

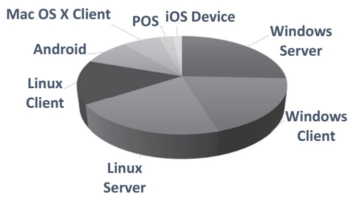

Figure 2: Distribution of Exploits with respect to platform.

identical products being sold on more than one market usually by the same seller (using an identical online handle). The products targeted 21 specific platforms, such as different versions of Adobe Flash, Linux, MSWindows and OS X as well as online presences such as Facebook, Wordpress and others. Hardware-related software such as those associated with Point-of-Sale machines, routers, and servers are also reflected in this number. Figure 2 illustrates the variety of products in the markets and Table 2 illustrates exemplar exploits.

Prod.	Vuln.	Target	USD
Kernel Panic	X-display system	Linux $<=$ 3.13.0-48	$\$471.56$
IE $<=$ 11	memory corr.	IE on Windows $<=$ 7	$\$35.00$
RemoteShell	wpconfig.php	Wordpress MU	$\$1,500$
0day RCE	WebView memory corr.	Android 4.1, 4.2	$\$36.50$
WindowsLPE	win32k elev. of priv.	Windows $<=$ 8.1	$\$12$-$48$
MS15-034 RCE	http.sys	Windows $<=$ 8.1	$\$311.97$
FUD Flash Exp.	unspec.	FlashPlayer $<=$16.0.0.287	$\$600.00$

Table 2: Examples of Exploits from Darknet Markets

System Configurations. As noted in Figure 2, a variety of platforms were represented in our darknet market data. In this paper, we describe results when using application constraints based on common configurations for Windows and Linux servers - as these were the most prominent targets of exploits found on the darknet. In our experiments, we mapped software such as media players, databases, and FTP server software to application constraint sets to model the functional requirements of a system. We have also created (and conducted experiments with) models for Android, Point-of-Sale, and Apple systems – though qualitatively the results differed little from the Windows and Linux Server experiments.

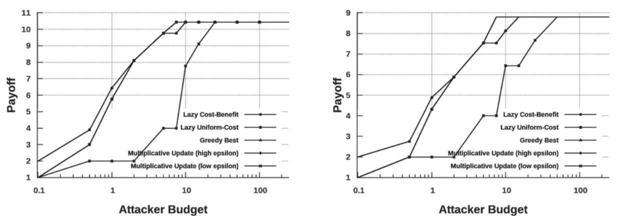

Figure 3: DHAP Payoff vs Budget - Left: Windows Server; Right: Linux Server.

DHAP Results. We implemented both the greedy and multiplicative update approaches to the DHAP problem. For the greedy algorithm, we studied three variants of greedy (cost-benefit, uniform cost, and combination of the the two) while we varied the parameter ϵ for the multiplicative update approach. We examined attacker payoff as a function of budget (in Bitcoin). Figure 3 displays this result. Though the cost-benefit greedy algorithm has the potential to perform poorly, it was, in general, the best performing approach - despite the multiplicative update approach achieving the better approximation guarantee. Further, the multiplicative update algorithm (Algorithm 2) was consistently the slowest in terms of runtime, taking much longer than the lazy greedy algorithms, particularly for high values of k_{atk}. Despite the multiplicative update algorithm having a better theoretical approximation ratio when compared to the tandem of greedy algorithms, namely $(1-\epsilon)(1-1/e)$ compared to $\frac{1}{2}(1-1/e)$, we see in Figure 3 that the greedy algorithms performed as well as or better than the multiplicative update very consistently. In all algorithms, as expected, runtime grew with budget (not pictured) - though the relationship was not strict, as an increase in budget does not necessarily mean that more exploits will be selected. In our experiments (on a commodity computer equipped with a 3.49 GHz i7 CPU and 16 GB of memory), our runtimes never exceeded ten minutes.

DHDP Results. Figure 4 demonstrate a defender's best response to an attack strategy against a Windows Server and Linux Server, respectively, for varying values of k_{atk}. Though we see similar trends in Figure 3 as we do in Figure 4, we see that the payoff is generally lower, meaning that the defender can lower the expected payoff by enacting a best response strategy to an attack strategy produced by DHAP - which in our framework translates to fewer exploited vulnerabilities.

Exploit Payoff Analysis. Instead of altering the software that appears on the host system in an attempt to avoid exploits, such as in the best response approach, in exploit payoff analysis, the defender will identify which specific exploits are increasing the payoff the most, with a hope that the defender can reverse-engineer the exploit, or patch the vulnerability himself. To identify which exploits should be reverse-engineered, the defender first runs

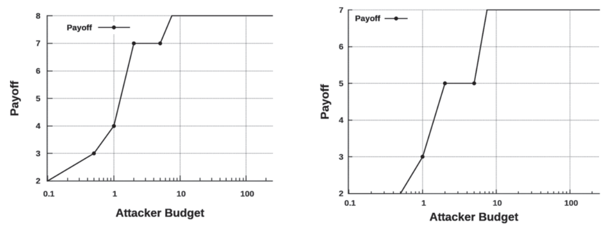

Figure 4: Defender Best Response, Payoff vs k_{atk} - Left: Windows Server; Right: Linux Server.

DHAP against his host system to identify what payoff an attacker could expect to produce. Then, for each exploit ex, the defender runs DHAP against the host with the set of exploits $Ex\backslash\{ex\}$. The exploit ex that, when removed from the universe of exploits Ex, produces the largest drop in payoff for the attacker is the exploit that the defender should attempt to reverse-engineer. More formally, let A be the attack strategy produced by DHAP when using Ex as the universe of exploits and let A_{ex} be the attack strategy that is produced when DHAP is run against the host when using $Ex\backslash\{ex\}$ as the universe of exploits. The defender will attempt to reverse-engineer the exploit $ex = \mathrm{argmax}_{ex\in Ex}\, p(A, D) - p(A_{ex}, D)$, where D is the defense strategy representing the host. To account for exploits that, though they greatly reduce payoff when removed from Ex, may be too expensive for the defender to purchase, we also consider a cost-benefit analysis, where the decrease in payoff is normalized by the cost of the exploit (i.e. $ex = \mathrm{argmax}_{ex\in Ex} \frac{p(A,D)-p(A_{ex},D)}{c_{ex}}$). The top exploits to reverse-engineer to defend a Windows Server host when considering an attacker budget of $k_{atk} = 5$, are shown in Table 3 with columns for both maximum payoff reduction and maximum cost-benefit analysis.

Exploit	Payoff Reduction	Max. Cost-Benefit	Exploit Cost (BTC)
SMTP Mail Cracker	1	4.757	0.2102
SUPEE-5433	1	1.190	0.8404
Hack ICQ	1	79.089	0.01264
Plasma	0.6677	1.582	0.2563
Wordpress Exploiter	0.6677	2.6467	0.2102
CVE-2014-0160	0.6677	3.178	0.2101

Table 3: Defender Exploit Analysis for $k_{atk} = 5$

Discussion. In future work, we plan to extend the game-theoretic framework to include non-deterministic problem formulations and construct algorithms to generate mixed strategies for the attacker and defender. By extending the exploit function in the framework, we plan to support blended threats, where the number of vulnerabilities affected by a cyber-attack is a superset of the union of the vulnerabilities affected by each individual exploit (i.e. $ExF(A) \supseteq \bigcup_{a\in A} ExF(\{a\})$).

Acknowledgements This work was supported by ASU GSI and the ONR Neptune program.

References

Ablon, L.; Libicki, M. C.; and Golay, A. A. 2014. *Markets for Cybercrime Tools and Stolen Data: Hackers' Bazaar*. Rand Corporation.

Azar, Y., and Gamzu, I. 2012. Efficient submodular function maximization under linear packing constraints. *ICALP* 1:38–50.

Chen, H. 2011. *Dark web: Exploring and data mining the dark side of the web*, volume 30. Springer Science & Business Media.

Feige, U. 1998. A threshold of ln n for approximating set cover. *J. ACM* 45(4):634–652.

Kiekintveld, C.; Lisý, V.; and Píbil, R. 2015. Game-theoretic foundations for the strategic use of honeypots in network security. In *Cyber Warfare - Building the Scientific Foundation*. 81–101.

Leskovec, J.; Krause, A.; Guestrin, C.; Faloutsos, C.; VanBriesen, J.; and Glance, N. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 420–429. ACM.

Li, W., and Chen, H. 2014. Identifying top sellers in underground economy using deep learning-based sentiment analysis. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, 64–67.

Lye, K.-w., and Wing, J. M. 2005. Game strategies in network security. *International Journal of Information Security* 4(1):71–86.

Minoux, M. 1978. Accelerated greedy algorithms for maximizing submodular set functions. In Stoer, J., ed., *Optimization Techniques*, volume 7 of *Lecture Notes in Control and Information Sciences*. Springer Berlin Heidelberg. 234–243.

Nemhauser, G.; Wolsey, L.; and Fisher, M. 1978. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming* 14(1):265–294.

Nguyen, K. C.; Alpcan, T.; and Başar, T. 2009. Stochastic games for security in networks with interdependent nodes. In *Game Theory for Networks, 2009. GameNets' 09. International Conference on*, 697–703. IEEE.

Shakarian, P., and Shakarian, J. 2015. Considerations for the development of threat prediction in the cyber domain. *submitted*.

Shakarian, P.; Shakarian, J.; and Ruef, A. 2013. *Introduction to cyber-warfare: A multidisciplinary approach*. Elsevier.

Soska, K., and Christin, N. 2015. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *24th USENIX Security Symposium (USENIX Security 15)*, 33–48. Washington, D.C.: USENIX Association.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. New York, NY, USA: Cambridge University Press, 1st edition.

Zhao, Z.; Ahn, G.-J.; Hu, H.; and Mahi, D. 2012. Socialimpact: Systematic analysis of underground social dynamics. In Foresti, S.; Yung, M.; and Martinelli, F., eds., *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, 877–894. Springer.