Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Ссылка на статью:
// Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2016. № 8. С. 182–201.
DOI: **10.7463/0816.0844738**
Представлена в редакцию: 29.06.2016

УДК 004.492.3

Подходы к созданию самонастраивающихся высокоинтерактивных клиентских обманных систем

Артюшкин А. С.[1,*]

[*]an.artyushkin@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

Статья посвящена методу автоматической оптимизации высокоинтерактивных клиентских обманных систем. Рассмотрена зависимость возможности обнаружения вредоносного поведения от основных характеристик клиентской обманной системы: количества ловушек, набора версий установленного потенциально уязвимого программного обеспечения, точек выхода во внешнюю сеть. Предложен метод поиска оптимальных параметров обманной системы. Метод опробован на клиентской обманной системе с открытым исходным кодом. Полученные результаты подтверждают применимость предложенного метода в промышленных средствах динамического анализа приложений. В дальнейшем планируется использовать оптимизируемые по предложенному методу высокоинтерактивные клиентские обманные системы для массового сканирования потенциально зараженных интернет-ресурсов.

Ключевые слова: оптимизация; динамический анализ; эксплойты; обманные системы; зараженные интернет-ресурсы

Введение

В последние годы активно изучается использование обманных систем (англ. honeypots) для обнаружения атак [2, 8]. Такие системы представляют собой наборы изолированных виртуальных или физических машин, являющихся приманкой для злоумышленников. Хакерские атаки на веб-серверы анализируются с помощью серверных обманных систем (server-side honeypots), а попытки компрометации компьютеров пользователей с помощью клиентских обманных систем (client-side honeypots). Клиентские обманные системы разделяются на низкоинтерактивные и высокоинтерактивные в зависимости от глубины симуляции реального окружения на ловушках в этих системах [5]. Обманные системы являются средствами динамического анализа. На основе фиксируемых в них событий могут срабатывать поведенческие правила. Вердикт об обнаружении вредоносной активности выдается по результатам срабатывания поведенческих правил.

В данной статье рассматриваются методы оптимизации клиентских высокоинтерактивных обманных систем на примере Cuckoo Sandbox (*https://www.cuckoosandbox.org/*). Проект

Cuckoo Sandbox представляет собой песочницу, позволяющую автоматически анализировать в виртуальных машинах файлы и интернет-ресурсы. При использовании Cuckoo ссылки на веб-ресурсы обходятся в ловушках полнофункциональным браузером. При таком сценарии работы Cuckoo Sandbox можно считать высокоинтерактивной клиентской обманной системой. Другие клиентские обманные системы ориентированные на анализ зараженных сайтов и песочницы перечислены в работах [7, 11]. Среди них Cuckoo Sandbox выделяется тем, что может быть использована как высокоинтерактивная обманная система, ее исходные коды открыты и она позволяет анализировать как файлы так и интернет-ресурсы. В этой статье песочница Cuckoo Sandbox будет рассматриваться как высокоинтерактивная клиентская обманная система.

Одним из способов распространения вредоносных программ является подгрузка исполняемых файлов через браузер с использованием эксплойтов. При посещении зараженного сайта эксплойты могут выполниться или не сработать в зависимости от версии браузера, установленных плагинов, IP-адреса и предполагаемого географического положения атакуемого компьютера [9, 10]. Для выявления поведения исполняемых файлов также важны версии программного обеспечения, которыми они запускаются в обманной системе. Например, поведение PDF-эксплойтов зависит от версии Adobe Reader, а поведение зараженных офисных документов — от версии MS Office.

Версии установленного на ловушках пользовательского программного обеспечения, IP-адрес и точка выхода во внешнюю сеть определяются задаваемой конфигурацией обманной системы. Как физические так и виртуальные ловушки могут иметь различный объем оперативной памяти, количество ядер процессора и требуемый объем места на жестком диске. Уязвимый софт различных версий может быть предустановлен на определенных снимках состояний ловушек либо установлен перед запуском задачи. Кроме этого при запуске задачи в большинстве случаев задается определяемый оператором интервал времени сканирования, по истечении которого задача будет принудительно завершена. Если вредоносная активность не будет зафиксирована в течение заданного интервала времени, поведенческие правила не смогут сработать. Для обработки большого количества подозрительных объектов нужно определять оптимальный набор параметров задачи, при котором вредоносные программы могут выполниться в ловушках, а значит смогут сработать поведенческие правила.

При потоковой обработке требуется, чтобы обманная система обрабатывала задачи как можно быстрее и возвращала максимально точные результаты. Можно считать, что эффективность обманной системы зависит от числа правильно вынесенных вердиктов, возвращаемых за единицу времени. Настоящая работа посвящена разработке методов, позволяющих оптимизировать параметры клиентской обманной системы на основе результатов ранее выполненных на ней задач. Поскольку для оптимизации используются только результаты выполненных сканирований и автоматически получаемые данные, предложенные методы

могут лечь в основу алгоритмов построения клиентских обманных систем автоматически настраивающихся на известные виды атак.

В разделе 1 приводится постановка задачи. В разделе 2 рассматривается метод автоматического выбора VPN-шлюза, с которого ловушка выходит во внешнюю сеть. В разделе 3 рассматривается метод поиска уязвимых версий программного обеспечения, устанавливаемого на ловушках. В разделе 4 рассматривается метод поиска оптимального количества ловушек. В разделе 5 представлен метод оптимизации конфигураций обманных систем, опирающийся на результаты предыдущих разделов.

1. Постановка задачи

Пусть для установки клиентской обманной системы доступны p выходов во внешнюю сеть из различных регионов мира, n версий потенциально уязвимого программного обеспечения и компьютер, позволяющий развернуть и запускать до m ловушек одновременно. Пусть также доступен выбор интервала времени сканирования из d вариантов. Будем считать, что поведенческие правила написаны экспертами, протестированы и уточнены так, что вероятность ложных срабатываний этих правил минимальна. Нужно разработать метод запуска задач на ловушках обманной системы и анализа результатов их выполнения для поиска параметров, при которых эффективность обманной системы максимальна.

Эффективность обманной системы E будем рассчитывать как произведение скорости выполнения задач v на точность вынесения вердиктов ν:

$$E = v\nu. \tag{1}$$

Скорость выполнения задач определяется количеством задач N, выполняемых за единицу времени:

$$v(t) = N'(t). \tag{2}$$

Под точностью будем понимать вероятность вынесения правильного вердикта (вердикт считается правильным, если проанализированный вредоносный объект назван обманной системой опасным, а просканированный невредоносный объект — чистым):

$$\nu = P(TruePositive) + P(TrueNegative). \tag{3}$$

Полный перебор всех возможных комбинаций потребует $pnmd$ запусков задач. На нашем экспериментальном стенде с 36 выходами во внешнюю сеть, 200 версиями Adobe Flash Player, и максимально доступными 10 ловушками на компьютере с процессором core i7 6770k это потребовало бы 288000 запусков задач.

Заметим, что точность ν можно выразить через вероятности ложных срабатываний и вероятность пропуска вредоносного объекта:

$$\nu = 1 - P(FalsePositive) - P(FalseNegative). \tag{4}$$

По условию, в клиентской обманной системе есть поведенческие правила, вероятность ложных срабатываний которых сведена к минимуму. Вероятность ложных срабатываний не зависит от конфигурации обманной системы и в данной работе не рассматривается:

$$P(FalsePositive) \to 0 \Rightarrow \nu \to 1 - P(FalseNegative). \quad (5)$$

Вероятность пропуска вредосного файла зависит от наличия в логах обманной системы вредоносного поведения. Если вредоносный файл не может выполниться на ловушке, то $P(FalseNegative) \to 1$ и эффективность обманной системы снижается.

Скорость работы обманной системы желательно повышать. Если скорость обработки v мала, то невозможно анализировать большие потоки файлов. Для обработки же небольших объемов файлов может быть целесообразнее привлекать экспертов.

В общем случае задачу, выполняемую в высокоинтерактивной клиентской обманной системе можно разбить на три этапа: доукомплектация ловушки необходимым для анализа программным обеспечением, обращение к подозрительному ресурсу и загрузка выложенных на нем файлов, запуск и выполнение файлов на ловушке.

2. Поиск оптимального выхода во внешнюю сеть

В ряде случаев прежде чем начать эксплуатацию браузера вредоносный код на зараженном сайте проверяет IP-адрес пользователя и его предполагаемое географическое положение [9, 10]. Это связано как с нежеланием злоумышленников оставлять следы атак в обманных системах антивирусных компаний так и с направленностью атак на определенную группу пользователей. Согласно исследованию [10] при заходе с подходящих IP-адресов на сайты зараженные набором эксплойтов Neutrino браузеру будет отправлен специальный iframe со ссылкой на эксплойт. Будем считать, что обманная система умеет автоматически определять наличие такого элемента в HTML-странице, который однозначно идентифицирует поведение эксплойтов Neutrino и требуется только подобрать подходящий VPN-шлюз на ловушке.

Выбор VPN-шлюза должен обеспечивать как обход проверки на IP-адрес так и наименьшее время последующей загрузки вредоносных файлов. Можно предположить, что предполагаемые географические расположения атакуемых машин и зараженных сайтов должны быть близки, чтобы проходить проверки в зараженном коде. В таком случае полезно оценивать географическое положение анализируемых обманной системой интернет-ресурсов. В работе [4] проводится анализ атак на обманные системы с использованием геолокации с точностью до города.

В данной работе был опробован метод геолокации интернет ресурса с помощью утилиты ping. Такой метод определения географического положения был предложен в работе [3]. Были рассчитаны задержки отклика от ловушек до шлюзов VPN и от ловушек до целевого сервера через VPN, используя ping. ICMP-запрос проходя через шлюз, попадает на целевой сервер и возвращается через шлюз обратно. По предположению ближайшим к целевому

серверу является шлюз, у к оторого наименьшая разница времени отклика до цели и времени отклика до самого шлюза.

Поскольку в рамках исследования не удалось найти активный сайт с работающим набором эксплойтов, эксперимент по поиску оптимального VPN-шлюза рассмотрен на примере анализа прямой ссылки на исполняемый файл. Для исследования была использована популярная VPN-сеть с 36 доступными выходами в интернет. Требовалось определить географическое положение известного ресурса. В качестве примера была выбрана прямая ссылка на утилиту Putty:

https://the.earth.li/sgtatham/putty/latest/x86/putty.exe

Целью эксперимента была проверка гипотезы о том, что время загрузки файла с интернет-ресурса коррелирует со временем отклика сервера на ICMP-запрос. Также требовалось узнать насколько сильно время загрузки файла зависит от региона, в котором географически расположен VPN-шлюз, через который осуществляется доступ к интернет-ресурсу. Для оценки времени загрузки Putty.exe была использована утилита Wget. Эксперимент проводился на виртуальных машинах VirtualBox с предустановленной операционной системой Windows 7 и VPN-клиентом Private Internet Access. Шаги эксперимента были следующие:

1) переключить VPN-клиент на один из p доступных VPN-шлюзов;

2) определить IP-адрес VPN-шлюза в графическом интерфейсе либо с помощью внешних сервисов по определению IP;

3) с помощью команды *ping «ip»* определить задержку отклика до выбранного VPN-шлюза;

4) с помощью команды *ping «domain»* определить задержку отклика до домена, на котором расположен целевой ресурс;

5) вычислить Δ как разницу между задержкой отклика до VPN-шлюза и задержкой отклика до домена с целевым ресурсом;

6) с помощью команды *wget «url»* скачать файл с целевого ресурса и определить необходимое время на его загрузку;

7) повторить предыдущие шаги для следующего доступного VPN-шлюза.

В табл. 1 указаны задержки ICMP-отклика до ссылки на Putty.exe через выходы VPN из разных регионов мира. По результатам проверки ближайшими к анализируемому ресурсу оказались VPN-шлюзы в Великобритании, Франции и Голландии.

Зависимость времени загрузки putty.exe от времени отклика указана на графике (рис. 1). По экспериментальным данным, время требуемое для загрузки файла по ссылке u может быть расчитано по задержке отклика t_{ping} через VPN-шлюз x и размеру файла $\text{size}(u)$:

$$t_{\text{download}}(x, u) = \frac{\text{size}(u) \cdot t_{\text{ping}}(x, u)}{50}. \tag{6}$$

Таблица 1

Задержки ICMP-отклика до ссылки на Putty.exe

i	Geolocation(x)	$t_{\text{ping}}(x,x)$, мс	$t_{\text{ping}}(x,u)$, мс	Δ
1	US California	210	355	145
2	US East	119	199	80
3	US Midwest	149	237	88
4	US Texas	180	295	115
5	US Florida	159	260	101
6	US Seattle	208	367	159
7	US West	196	338	142
8	US Silicon Valley	199	349	150
9	US New York City	142	220	78
10	**UK London**	**58**	**73**	**15**
11	**UK Southampton**	**55**	**68**	**13**
12	CA Toronto	157	245	88
13	CA North York	132	230	98
14	AU Sydney	317	625	308
15	AU Melbourne	332	601	269
16	New Zealand	317	578	261
17	**Netherlands**	**55**	**69**	**14**
18	Sweden	24	62	38
19	Norway	75	96	21
20	Denmark	53	72	19
21	Finland	32	86	54
22	Switzerland	58	81	23
23	**France**	**68**	**76**	**8**
24	Germany	41	68	27
25	Ireland	72	103	31
26	Italy	71	104	33
27	Russia	11	68	57
28	Romania	96	156	60
29	Turkey	88	152	64
30	Hong Kong	210	432	222
31	Singapore	318	515	197
32	Japan	310	580	270
33	Israel	130	188	58
34	Mexico	202	351	149
35	Brazil	279	557	278

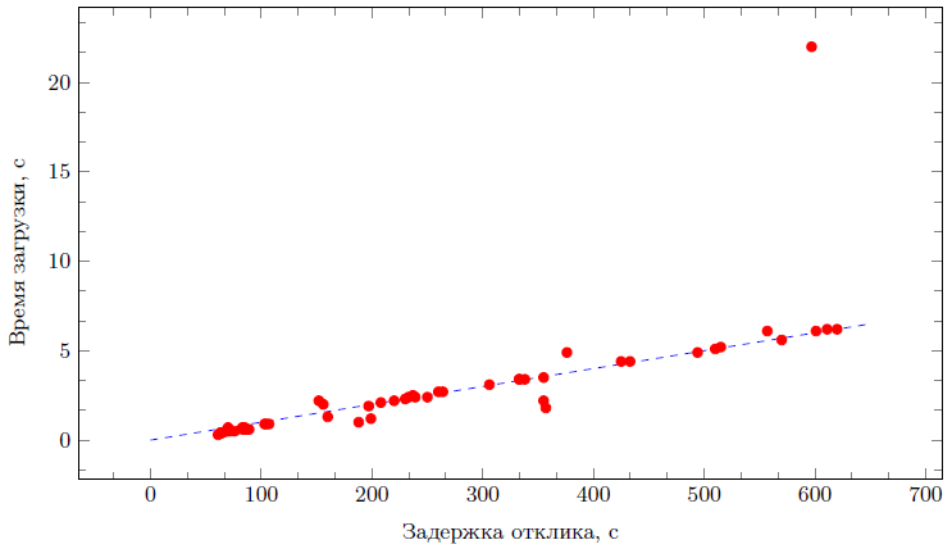

Рис. 1. Зависимость времени загрузки putty.exe от задержки отклика до выбранного шлюза

Предлагаемый метод поиска оптимального выхода во внешнюю сеть с ловушек состоит в следующем:

1) отправить ICMP-запросы серверу, на котором расположен анализируемый интернет-ресурс, со всех p возможных VPN-шлюзов и получить задержки отклика;

2) отсортировать доступные выходы во внешнюю сеть по возрастанию задержки отклика;

3) запустить задачи по сканированию зараженного интернет-ресурса, выбирая шлюзы из отсортированного списка, до тех пор, пока не сработает правило на появление в трафике признаков подгрузки вредоносного HTML-элемента;

4) сохранить выбор шлюза, полученного на предыдущем шаге, для использования на последующих этапах.

В худшем случае подтребуется проверить все возможные VPN-шлюзы. Для этого потребуется запустить p задач на ловушках с интервалом сканирования достаточным для подгрузки контента с анализируемого интернет ресурса. Версия потенциально уязвимого плагина выбирается произвольно, поскольку срабатывание правила на появление первых фрагментов вредоносного кода от нее не зависит. Таким образом, на поиск оптимального выхода во внешнюю сеть потребуется $\mathcal{O}(p)$ запусков задач.

Таким образом, за $\mathcal{O}(p)$ запусков задач можно подобрать VPN-шлюзы, с которых при заходе на подозрительный сайт осуществляется попытка атаки на ловушку. Для каждого из таких шлюзов по предполагаемому объему загружаемого контента и задержке ICMP-отклика можно оценить время загрузки вредоносного контента в ловушку.

3. Поиск уязвимых версий программного обеспечения

В плагинах к браузеру ежегодно находят новые уязвимости. После опубликования информации об уязвимости производитель плагина, как правило, выпускает новую версию плагина с патчем, устраняющим данную уязвимость. Новые версии плагинов выходят регулярно и

их может быть достаточно много. Например, на сайте Adobe доступны для скачивания более двухсот версий Adobe Flash Player, который часто является целевым плагином для эксплойт-паков.

Поскольку об уязвимости в конкретной версии плагина не всегда становится известно сразу, она может присутствовать в нескольких подряд выпущенных версиях плагина. Предположение, что существует набор подряд идущих версий плагина, содержащих одну и ту же уязвимость, позволяет упростить поиск конфигурации софта, необходимой для запуска эксплойта.

Пусть имеется выбор из n версий уязвимого плагина и пусть среди них k версий подряд содержат уязвимость, которую использует анализируемый эксплойт. Требуется подобрать уязвимую версию и узнать диапазон уязвимых версий.

Очевидно, что если известна хотя бы одна версия плагина, при которой осуществляется успешная эксплуатация, то границы уязвимого диапазона можно определить бинарным поиском. Потребуется перебрать не более $\log_2 n$ версий старше уязвимой и не более $\log_2 n$ версий младше уязвимой. Остается найти оптимальный способ поиска первой уязвимой версии.

Можно предложить следующий алгоритм поиска уязвимой версии среди n возможных. Сначала проверим среднюю версию, т.е. версию с номером $\left\lfloor \frac{n}{2} \right\rfloor$. Затем в случае неуспеха, проверим версии с номерами $\left\lfloor \frac{n}{4} \right\rfloor$ и $\left\lfloor \frac{3n}{4} \right\rfloor$. Аналогично, в случае, если уязвимая версия не найдена будем проверять версии, находящихся в середине интервалов версий, ограниченных проверенными версиями. На последнем этапе будут проверяться $\frac{n}{2}$ версий. Алгоритм либо остановится на первой встреченной уязвимой версии и вернет ее номер, либо покажет, что уязвимых версий среди всех доступных не найдено. Работа алгоритма при $n = 10$ показана на рис. 2, а при $n = 16$ — на рис. 3. Нетрудно доказать, что алгоритм имеет сложность $\mathcal{O}\left(\frac{n}{k}\right)$.

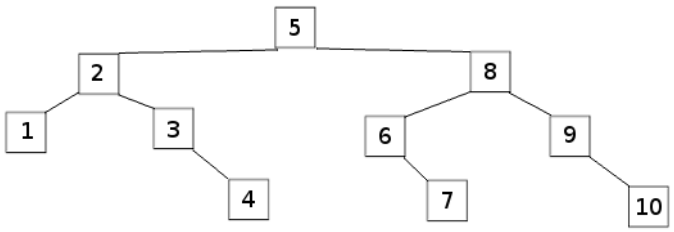

Рис. 2. Пример работы алгоритма поиска уязвимых версий для $n = 10$

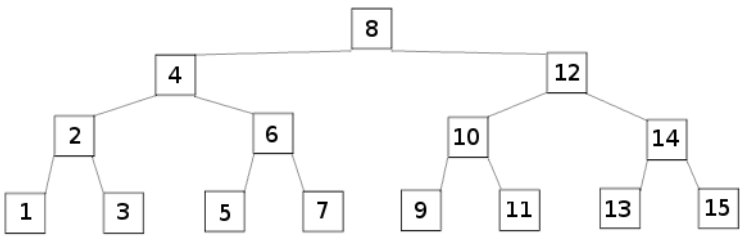

Рис. 3. Пример работы алгоритма поиска уязвимых версий для $n = 16$

Таким образом, сложность поиска уязвимого диапазона версий асимптотически равна $\mathcal{O}\left(\frac{n}{k} + \log_2 n\right)$. Если уязвимость была быстро исправлена и затронула небольшое число версий, число шагов алгоритма стремится к $\frac{n}{k}$. Если k велико, т.е. уязвимость характерна для большого числа версий, число шагов алгоритма стремится к $\log_2 n$.

Для проверки этой гипотезы был проведен эксперимент на виртуальной машине VirtualBox с установленной гостевой операционной системой Windows 7. В качестве зараженного ресурса был использован локально развернутый сервер apache, на котором был выложен SWF-эксплойт (*md5:e17cc5e55dc831bb07780609a4f669269*) из набора эксплойтов Neutrino. Задачи по сканированию запускались с помощью Cuckoo Sandbox 2.0 (*https://www.cuckoosandbox.org/*). В случае успешной эксплуатации этот вредоносный SWF-файл запускает на ловушке процесс *svchost.exe*. Эксперимент состоял из следующих шагов:

1) выбрать версию плагина Adobe Flash Player для браузера Internet Explorer согласно приведенному выше алгоритму;

2) установить выбранную версию на ловушку;

3) запустить сканирование зараженного локального ресурса с помощью Cuckoo Sandbox командой *submit.py –url «testurl»*;

4) после окончания сканирования в отчете Cuckoo Sandbox по выполненной задаче найти событие запуска процесса *svchost.exe*;

5) если запуск *svchost.exe* зафиксирован в отчете, считаем, что уязвимая версия найдена, иначе выбираем следующую версию по предложенному алгоритму;

6) после нахождения первой уязвимой версии ищем диапазон уязвимых версий, также ожидая появления в отчете по задаче появления события запуска процесса *svchost.exe*.

С помощью предложенного подхода удалось за 13 шагов установить уязвимые все версии Adobe Flash Player для вредоносного SWF-файла из набора эксплойтов Neutrino, md5 хеш которого равен *e17cc5e55dc831bb07780609a4f669269*.

Найдя диапазоны уязвимых версий для известных эксплойтов, можно выделить пересечения диапазонов. В таком случае для проверки подозрительного сайта на наличие известных эксплойтов понадобятся ловушки с версиями из найденных пересечений. Это позволит не тратить время на установку плагинов непосредственно перед запуском сканирования и число запусков ловушек с различными комбинациями софта.

Таким образом, алгоритм поиска уязвимых версий состоит в следующем:

1) выбираем медиану списка версий потенциально уязвимого программного обеспечения и запускаем задачу по сканированию зараженного интернет-ресурса с выбранной версией;

2) если вредоносное поведение не было зафиксировано, разбиваем список доступных версий на списки версий младше и старше версии, выбранной на предыдущем шаге; запускаем сканирования, используя версии-медианы полученных списков;

3) запускаем сканирования зараженного интернет-ресурса, выбирая медианы списков версий ограниченных выбранными ранее версиями до тех пор, пока не будет зафиксировано вредоносное поведение либо не будут проверены все доступные версии;

4) если была найдена хотя бы одна уязвимая версия, определяем диапазон уязвимых версий с помощью бинарного поиска;

5) сохраняем диапазон уязвимых версий.

На этом этапе потребуется $\mathcal{O}\left(\frac{n}{k} + \log_2 n\right)$ запусков задач. В худшем случае придется перебрать все n версий плагина, чтобы найти единственную уязвимую или определить, что уязвимых версий нет среди доступных.

4. Оценка зависимости времени выполнения задач от количества ловушек

На данном этапе считаем, что на ловушках обманной системы подобран подходящий выход в интернет и установлены нужные для работы некоторого эксплойта версии программного обеспечения. Остается найти оптимальное число виртуальных машин z, которые будут использованы в качестве ловушек. Количество ловушек ограничивается ресурсами компьютера, на котором развернута обманная система. Минимальный объем оперативной памяти и места на жестком диске зависит от операционной системы, устанавливаемой на ловушке.

Исследование было проведено на компьютере следующей конфигурации: *core i7 6770k, 8Gb RAM, 960Gb SSD*. В качестве обманной системы использовалась *Cuckoo Sandbox 2.0*. На ловушках была установлена операционная система *Windows 7*. Анализировались ссылки на зараженный *SWF-файл* из набора эксплойтов *Neutrino*, упомянутого в предыдущем разделе. Полезной нагрузкой эксплойта был троян-шифровальщик. Целью исследования было установить оптимальное число ловушек.

В первую очередь опытным путем была определено как изменяется производительность песочницы v в зависимости от количества виртульных машин. Было посчитано время T_z необходимое для выполнения ста задач по сканированию ссылки на эксплойт. Интервал сканирования для каждой задачи был задан равным 20 с. Суммарное время сканирования всех задач в зависимости от числа ловушек указано на графике ниже синим цветом. Красным цветом отображена точность работы песочницы. Критерием успешного обнаружения вредоносного поведения было выбрано появление файла svchost.exe в каталоге временных файлов.

Производительность v, т.е. скорость выполнения задач в единицу времени пропорциональна числу ловушек, поскольку задачи выполняются на них параллельно. При независимо параллельно работающих ловушках можно считать, что увеличение количества ловушек в несколько раз, пропорционально увеличит эффективность E. Но в нашем случае при увеличении числа ловушек резко падала точность работы каждой из них. Это связано с тем, что чем больше ресурсов выделялось виртуальным машинам, тем меньше оставалось ядру

системы. В результате при заданном интервале сканирования в 20 с наиболее эффективной оказалась песочница всего лишь с двумя ловушками. Зависимость времени сканирования от числа ловушек показана на рис. 4.

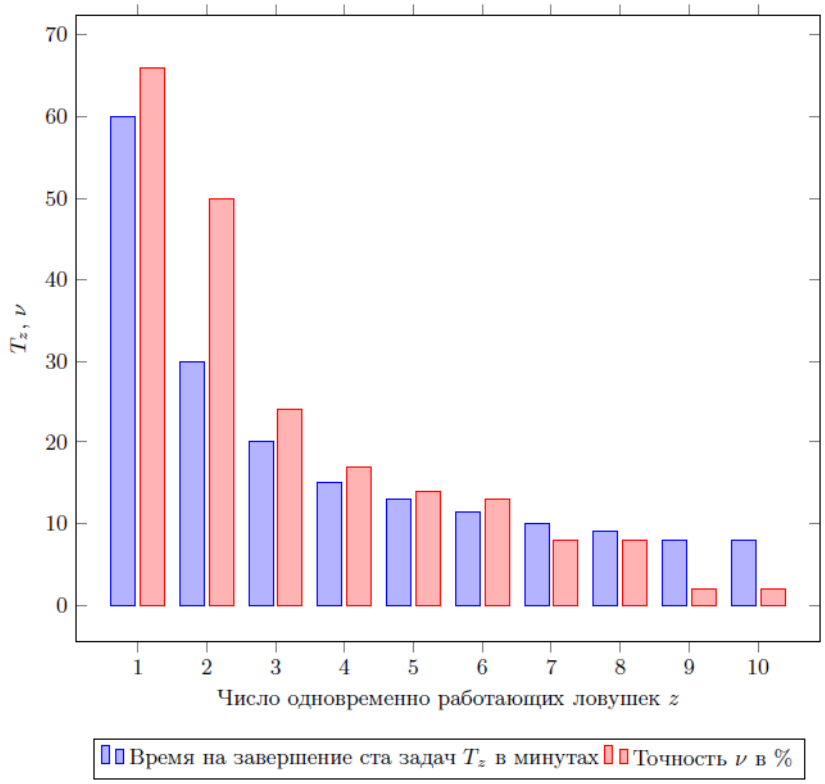

Рис. 4. Зависимость времени сканирования и точности системы от числа ловушек

Потребовалось найти способ нахождения оптимальной комбинации времени сканирования t и числа виртуальных машин z, при которых эффективность обманной системы максимальна. В первую очередь было решено найти все комбинации t и z, при которых достигается заданная точность ν, а затем среди них выбрать вариант с наименьшим временем сканирования T_z.

Точность ν — функция, действующая на множестве пар z и t в процент задач, в которых вредоносное поведение было зафиксировано и сработали поведенческие правила. По предположению, ν возрастает по t и убывает по z. Иными словами чем больше времени дается на выполнение анализа и чем меньше ресурсов тратися на запуск виртуальных машин, тем вероятнее, что выполнится вредоносный файл и сработают правила. В первую очередь требуется найти все пары z и t, для которых ν достигает минимально допустимых пороговых значений.

В предположении, что ν возрастает по t и убывает по z, для поиска пар (z, t), удовлетворяющих заданному критерию точности можно воспользоваться алгоритмом седлового поиска [1]. При постановке задачи в разделе 2 было задано максимальное количество виртульных машин m и фиксированное число вариантов выбора времени сканирования d. Седловой

поиск позволяет вместо полного перебора $m \cdot d$ комбинаций находить все искомые пары (z, t) за $O(m + d)$ операций.

Седловой поиск начинается на комбинации $(t_{\min}, z_{\min})$. На каждом этапе область поиска ограничивается прямоугольником с левым нижним углом в точке (t, z) и правым верхним углом в точке $(t_{\max}, z_{\max})$. Если точность ν в точке (t, z) меньше заданного порога, проверяем точность работы системы в точке $(t + 1, z)$. Если требуемая точность будет достигнута на некотором выборе t', проверим работу системы в точке $(t', z + 1)$. Алгоритм прекращает работу, если значения t или z становятся больше допустимых. На каждом шаге алгоритма мы либо увеличиваем t либо z. В худшем случае алгоритм обходит периметр прямоугольника от левого нижнего угла до правого верхнего, что потребует проверок $d + m$ конфигураций.

Гипотеза применимости седлового поиска для поиска оптимальной конфигурации была проверена на экспериментальном стенде. Оказалось, что условия возрастания точности ν соблюдаются не во всех случаях. На основе результатов выполнения ста задач по сканированию вредоносного SWF-файла из набора эксплойтов Neutrino были выбраны три поведенческих правила. Точность была посчитана отдельно для каждого из трех выбранных правил:

- появление файла svchost.exe в каталоге временных файлов;
- появление в песочнице более тысячи зашифрованных файлов;
- появление в песочнице более двухсот зашифрованных файлов в среднем на каждой ловушке.

Были проведены эксперименты на ловушках, управляемых Cuckoo Sandbox для проверки гипотезы применимости седлового поиска для оптимизации количества ловушек по производительности и точности. Эксперимент состоял из следующих шагов:

1) в конфигурационном файле cuckoo.conf выбрать требуемое количество ловушек;

2) запустить сто задач по сканированию локально ресурса с SWF-эксплойтом скриптом, состоящем из ста команд *submit.py –url «testurl» –timeout «t»*, где t определяемый по приведенному выше алгоритму интервал сканирования;

3) после окончания сканирования в отчете Cuckoo Sandbox по выполненной задаче найти событие запуска процесса *svchost.exe* и количество файлов с расширением *crypt*;

4) повторить предыдущие шаги для 2, 4, 6, 8, 10 ловушек и интервалов времени 20, 40, 60, 80 секунд.

Эксперименты показали, что искать оптимальную конфигурацию седловым поиском, используя первое правило нельзя. Число появлений файла svchost.exe при увеличении времени сканирования возрастает до определенного порогового значения t, а затем резко убывает. Это связано с тем, что при увеличении t выше определенного порога, успевает начать работу подгружаемый эксплойтом троян-шифровальщик. Загруженный троян начинает шифровать файлы на ловушках. Чем больше файлов зашифровано, тем выше нагрузка на ядро обманной системы и тем меньше ресурсов остается на выполнение задач в других ловушках. Как следствие, чем активнее шифруются файлы, тем сильнее падает точность обманной системы.

В табл. 2 выделены комбинации, при которых файл svchost.exe создан более чем в сорока процентах случаев. На каждую комбинацию z и t было проведено сто запусков задач. В ячейках таблицы указаны затраченное время и число задач, в которых зафиксировано появление svchost.exe.

Таблица 2

Число задач, в которых было обнаружено создание файла svchost.exe

Число ловушек	Интервал сканирования, с			
	20	40	60	80
10 VM	(8, 2)	(16, 30)	(**24**, **56**)	(28, 60)
8 VM	(10, 8)	(14, 17)	(**22**, **64**)	(30, 46)
6 VM	(13, 13)	(**22**, **54**)	(35, 38)	(42, 25)
4 VM	(15, 17)	(**24**, **48**)	(38, 44)	(62, 25)
2 VM	(**28**, **46**)	(26, 42)	(81, 46)	(130, 46)

В табл. 3 выделены комбинации, при которых в обманной системе появилось более тысячи зашифрованных файлов на всех ловушках. На каждую комбинацию z и t было проведено сто запусков задач. Число зашифрованных файлов возрастает по t и убывает по z. Выделенные комбинации можно было получить седловым поиском. В ячейках таблицы указаны затраченное время и число зашифрованных файлов на всех ловушках.

Таблица 3

Количество зашифрованных файлов на всех ловушках

Число ловушек	Интервал сканирования, с			
	20	40	60	80
10 VM	(8, 0)	(16, 0)	(24, 0)	(**28**, **23612**)
8 VM	(10, 0)	(14, 104)	(**22**, **7252**)	(30, 29426)
6 VM	(13, 0)	(22, 502)	(**35**, **8538**)	(42, 57625)
4 VM	(15, 0)	(**24**, **1031**)	(38, 21502)	(62, 57678)
2 VM	(28, 0)	(**46**, **29368**)	(81, 61956)	(130, 66868)

В табл. 4 выделены комбинации, при которых в обманной системе появилось более двухста пятидесяти зашифрованных файлов в среднем на каждую ловушку. На каждую комбинацию z и t было проведено сто запусков задач. Среднее число зашифрованных файлов также возрастает по t и убывает по z. Выделенные комбинации можно было получить седловым поиском. В ячейках таблицы указаны затраченное время и число зашифрованных файлов в среднем на каждой ловушке.

По результатам экспериментов, общее количество зашифрованных файлов и среднее количество зашифрованных файлов на каждой ловушке возрастают по t и убывают по z. Если

Таблица 4

Среднее количество зашифрованных файлов на одной ловушке

Число ловушек	Интервал сканирования, с			
	20	40	60	80
10 VM	$(8, 0)$	$(16, 0)$	$(24, 0)$	$(\mathbf{28}, \mathbf{2361})$
8 VM	$(10, 0)$	$(14, 13)$	$(\mathbf{22}, \mathbf{906})$	$(30, 3678)$
6 VM	$(13, 0)$	$(22, 63)$	$(\mathbf{35}, \mathbf{1423})$	$(42, 9604)$
4 VM	$(15, 0)$	$(\mathbf{24}, \mathbf{257})$	$(38, 5375)$	$(62, 14421)$
2 VM	$(28, 0)$	$(\mathbf{46}, \mathbf{14984})$	$(81, 30978)$	$(130, 33434)$

клиентская обманная система при вынесении вердикта опирается на количество зашифрованных файлов, то для поиска конфигураций, при которых достигается заданная точность можно использовать седловой поиск. Среди найденных комбинаций затем можно выбрать вариант, при котором время, затраченное на все задачи минимально.

Таким образом с помощью седлового поиска можно находить пары (z, t), при которых достигается заданная точность работы обманной системы, можно за $\mathcal{O}(m + d)$ проверок конфигураций. Среди найденных пар (z, t) остается выбрать комбинацию с наименьшим общим временем сканирования.

5. Метод оптимизации параметров обманной системы

В предыдущих разделах были детально рассмотрены этапы оптимизации выхода во внешнюю сеть с ловушек обманной системы, версий уязвимого программного обеспечения и количества ловушек. Итоговый вариант метода оптимизации обманных систем опирается на результаты, представленные в предыдущих разделах.

Пусть известен адрес зараженного интернет-ресурса, на котором расположен активно работающий эксплойт и имеется клиентская обманная система, в которой можно изменять число ловушек, устанавливать потенциально уязвимое программное обеспечение и выбирать выход во внешнюю сеть. В таком случае для нахождения наиболее эффективной конфигурации обманной системы можно воспользоваться следующим методом.

1. Поиск оптимального выхода во внешнюю сеть:

а) отправить ICMP-запросы серверу, на котором расположен анализируемый интернет-ресурс, со всех доступных VPN-шлюзов и получить задержки отклика;

б) отсортировать доступные выходы во внешнюю сеть по возрастанию задержки отклика сервера;

в) запустить задачи по сканированию зараженного интернет-ресурса, выбирая шлюзы из отсортированного списка, до тех пор, пока не сработает правило на появление в трафике признаков подгрузки вредоносного HTML-элемента;

г) сохранить выбор шлюза, полученного на предыдущем шаге, для использования на последующих этапах.

2. Поиск уязвимых версий программного обеспечения:

а) выбираем медиану списка версий потенциально уязвимого программного обеспечения и запускаем задачу по сканированию зараженного интернет-ресурса с выбранной версией;

б) если вредоносное поведение не было зафиксировано, разбиваем список доступных версий на списки версий младше и страше версии, выбранной на предыдущем шаге; запускаем сканирования, используя версии-медианы полученных списков;

в) запускаем сканирования зараженного интернет-ресурса, выбирая медианы списков версий ограниченных выбранными ранее версиями до тех пор, пока не будет зафиксировано вредоносное поведение либо не будут проверены все доступные версии;

г) если была найдена хотя бы одна уязвимая версия, определяем диапазон уязвимых версий с помощью бинарного поиска;

д) сохраняем диапазон уязвимых версий.

3. Поиск оптимального количества ловушек обманной системы:

а) запускаем задачи на одной машине увеличивая интервал сканирования до тех пор, пока вредоносное поведение не начнет стабильно фиксироваться обманной системой. Сохраняем полученное допустимое значение интервала времени сканирования;

б) увеличиваем число машин на 1;

в) если перестали срабатывать поведенческие правила, увеличиваем интервал времени сканирования до тех пор, пока вредоносное поведение не начнет стабильно проявляться. Сохраняем полученное допустимое значение интервала времени сканирования;

г) повторяем предыдущие два шага, пока число ловушек не достигнет максимально возможного значения;

д) среди сохраненных допустимых пар значений числа ловушек и интервалов времени сканирования выбираем комбинацию, при которых общее время на выполнение задач минимально.

4. Сохранение выбранного выхода во внешнюю сеть, диапазона уязвимых версий и оптимального количества ловушек для проанилизированного зараженного интернет-ресурса.

При таком подходе параметры обманной системы выбираются поэтапно, так что на каждом последующем этапе фиксируется больше поведенческих признаков атаки на ловушку. Поведенческие признаки при анализе зараженного интернет-ресурса обычно связаны с проверкой версий плагинов, эксплуатацией браузера и запуском вредоносного исполняемого файла в ловушке. Попыток эксплуатации может не быть, если ip-адрес ловушки не удовлетворяет условиям в коде зараженного сайта, например, не принадлежит определенному географическому региону или находится в черном списке. Если не удалось подобрать подходящий выход во внешнюю сеть, атака на ловушку не начинаетя, и нет смысла перебирать остальные параметры обманной системы. Если же путем перебора выходов во внешнюю

сеть получилось добиться появления в сетевом трафике вредоносных скриптов или других вредоносных элементов интернет-страниц, продолжение атаки на ловушку может быть невозможно без правильно подобранных версий плагинов браузера. Эксплойты, как правило, используют уязвимости конкретных версий пользовательского программного обеспечения, в том числе плагинов браузера. Если на ловушке установлены уязвимые версии приложений, эксплуатация может привести к выполнению полезной нагрузки заложенной в эксплойт.

Стремление зафиксировать как можно больше стадий атаки на ловушки обманной системы обусловлено зависимостью точности вынесения вердиктов от собираемых поведенческих признаков. Наличие в сетевом трафике подозрительного элемента интернет-страницы не всегда однозначно говорит о попытке атаки на обманную систему. В случае, когда зафиксированы все стадии атаки через браузер, и без участия пользователя в ловушке начинает выполняться вредоносный исполняемый файл, вероятность ложного срабатывания близка к нулю. Высокоинтерактивные клиентские обманные системы гибко настраиваются и содержат большое количество параметров сканирования для создания условий успешной эксплуатации ловушек.

Предложенный метод позволяет находить комбинации параметров обманной системы, подходящие для выявления атак, быстрее чем полный перебор параметров. Необходимое число шагов для оптимизации по предложенному методу конечно и зависит от числа вариантов выбора каждого параметра. Этап выбора оптимального выхода с ловушек во внешнюю сеть в худшем случае выполняется за p запусков задач, где p - число доступных VPN-шлюзов. Этап по поиску уязвимых версий программного обеспечения в худшем случае выполняется за n запусков задач, где n число доступных версий. В предположении о непрерывности диапазонов уязвимых версий тот этап выполняется за $\mathcal{O}\left(\frac{n}{k} + \log_2 n\right)$, где k — длина непрерывной последовательности уязвимых версий. Этап поиска оптимального количества ловушек выполняется в худшем случае за $m + d$ запусков задач, где m — максимально возможное число ловушек, а d — число вариантов выбора интервала времени сканирования. Следовательно, в худшем случае на прохождения всех этапов предложенного метода оптимизации потребуется выполнить $p + n + m + d$ запусков задач.

Заключение

Высокоинтерактивные клиентские обманные системы предоставляют оператору большое количество параметров для их гибкой настройки. Ручной перебор параметров во многих случаях приводит к неоптимальной конфигурации обманной системы, что снижает точность обнаружения вредоносных программ или в разы увеличивает затраты времени и ресурсов на проведение сканирований. Предложенный в данной работе метод автоматической оптимизации ловушек в клиентской обманной системе позволяет повысить точность детектирования и снизить время выполнения задач. Этапы метода протестированы на эксперименталь-

ном стенде. Результаты указывают на потенциальную полезность метода для применения в промышленных системах такого рода. В дальнейшем планируется использовать автонастраиваемую систему для массовых сканирований интернет ресурсов. Предложенный метод позволяет не только оптимизировать параметры обманной системы, но и сделать это максимально автоматизированно.

Список литературы

1. Bird R.S. Improving Saddleback Search: A Lesson in Algorithm Design // In: Mathematics of Program Construction. Springer, 2006. P. 82–89 (ser. Lecture Notes in Computer Science; vol. 4014). DOI: 10.1007/11783596_8
2. Алейнов Ю.В., Саушкин И.Н. Об оптимальной конфигурации обманных систем в компьютерной сети предприятия // Вестн. Сам. гос. техн. ун-та. Сер. Физ.-мат. науки. 2013. № 4(33). С. 107–114. DOI: 10.14498/vsgtu1259
3. Muir J.A., van Oorschot P.C. Internet Geolocation and Evasion and Counterevasion // ACM Computing Surveys. 2009. Vol. 42, iss. 1. Art. no. 4. DOI: 10.1145/1592451.1592455
4. van Polen M.G.T., Moura G.C.M., Pras A. Finding and Analyzing Evil Cities on the Internet // In: Managing the Dynamics of Networks and Services. Springer, 2011. P. 38–48 (ser. Lecture Notes in Computer Science; vol. 6734). DOI: 10.1007/978-3-642-21484-4_4
5. Mirza M., Usman M., Biuk-Aghai R.P., Fong S. A Modular Approach for Implementation of Honeypots in Cyber Security // International Journal of Applied Engineering Research. 2016. Vol. 11, no. 8. P. 5446–5451. Режим доступа: http://www.ripublication.com/ijaer16/ijaerv11n8_15.pdf (дата обращения 10.08.2016).
6. Wang Y.-M., Beck D., Jiang X., Roussev R. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities // Microsoft Research. MSR-TR-2005-72. 2005. Режим доступа: http://research.microsoft.com/apps/pubs/default.aspx?id=70182 (дата обращения 20.05.2016).
7. Kaur S., Kaur H. Client Honeypot Based Malware Program Detection Embedded Into Web Pages // International Journal of Engineering Research and Applications. 2013. Vol. 3, iss. 6. P. 849–854. Режим доступа: http://www.ijera.com/papers/Vol3_issue6/EM36849854.pdf (дата обращения 20.05.2016).
8. Zeng Y.G., Coffey D., Viega J. How Vulnerable Are Unprotected Machines on the Internet? // In: Passive and Active Measurement. Springer, 2014. P. 224–234 (ser. Lecture Notes in Computer Science; vol. 8362). DOI: 10.1007/978-3-319-04918-2_22
9. Maio G.D., Kapravelos A., Shoshitaishvili Y., Kruegel Ch., Vigna G. PExy: The Other Side of Exploit Kits // In: Detection of Intrusions and Malware, and Vulnerability Assessment.

Springer, 2014. P. 132–151 (ser. Lecture Notes in Computer Science; vol. 8550). DOI: 10.1007/978-3-319-08509-8_8

10. Rocha L. Neutrino Exploit Kit Analysis and Threat Indicators. SANS Institute. InfoSec Reading Room. Режим доступа: https://www.sans.org/reading-room/whitepapers/detection/neutrino-exploit-kit-analysis-threat-indicators-36892 (дата обращения 25.05.2016).
11. Kirat D., Vigna G., Kruegel C. Barecloud: Bare-metal Analysis-based Evasive Malware Detection // 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association. San Diego, CA. 2014. P. 287–301. Режим доступа: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat (дата обращения 27.05.2016).

Science&Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2016, no. 8, pp. 182–201.
DOI: **10.7463/0816.0844738**
Received: 29.06.2016

Approaches to creating self-modifying high interaction client honeypots

Artyushkin A. S.[1,*]

[*]an.artyushkin@gmail.com

[1]Bauman Moscow State Technical University, Russia

Keywords: optimization, dynamical analysis, exploits, honeypots, infected web sites

This article considers the problem of behavioral malware detection using the high interaction client honeypots. The focus is on solving the problem of behavioral signature extraction to detect attacks from infected websites. Mentioned conditions appropriate for successful execution of modern browser exploits in traps of high interaction client honeypot. Introduced a concept of the client honeypot efficiency based on the honeypot performance and accuracy.

The optimization method of the honeypot system parameters has been suggested to improve the efficiency of malware detection. Considered a case of optimization of a honeypot system to detect known threats, for which there are behavioral rules already written by experts. The method is based on results of tasks scanning infected websites and automatically conducted data, so it can be used for creating self-modifying honeypots.

The article considers in detail a possibility for behavioral malware detection versus configuration of client honeypots. Analyzes the dependence of time for downloading files on the choice a VPN gateway from which the honeypot trap is connected to the external network. Stated results of using method of geo-location by delays of the response from the target server. Proposed algorithm of finding vulnerable software versions installed on the honeypot traps. The experimental results proved the using saddleback search algorithm to optimize the number of traps in honeypot system and the timeout of the scanning.

The method has been tested on the open source high interaction client honeypot. The experimental results proved that the proposed method would be useful in the enterprise tools of dynamic analysis of products. In the future it is planned to use the optimized honeypots for massive scanning of potentially infected websites

References

1. Bird R.S. Improving Saddleback Search: A Lesson in Algorithm Design. In: *Mathematics of Program Construction*. Springer, 2006, pp. 82–89 (ser. Lecture Notes in Computer Science; vol. 4014). DOI: 10.1007/11783596_8

2. Aleinov Yu.V., Saushkin I.N. Optimal honeynet configuration in enterprise computer networks. *Vestn. Samar. Gos. Tekhn. Univ. Ser. Fiz.-Mat. Nauki = Bulletin of the Samara State Technical Unversity, ser. Phys.-Math. Sciences*, 2013, no. 4(33), pp. 107–114. DOI: 10.14498/vsgtu1259 (in Russian)
3. Muir J.A., van Oorschot P.C. Internet Geolocation and Evasion and Counterevasion. *ACM Computing Surveys*, 2009, vol. 42, iss. 1, art. no. 4. DOI: 10.1145/1592451.1592455
4. van Polen M.G.T., Moura G.C.M., Pras A. Finding and Analyzing Evil Cities on the Internet. In: *Managing the Dynamics of Networks and Services*. Springer, 2011, pp. 38–48 (ser. Lecture Notes in Computer Science; vol. 6734). DOI: 10.1007/978-3-642-21484-4_4
5. Mirza M., Usman M., Biuk-Aghai R.P., Fong S. A Modular Approach for Implementation of Honeypots in Cyber Security. *International Journal of Applied Engineering Research*, 2016, vol. 11, no. 8, pp. 5446–5451. Available at: http://www.ripublication.com/ijaer16/ijaerv11n8_15.pdf accessed 10.08.2016.
6. Wang Y.-M., Beck D., Jiang X., Roussev R. *Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities*. Microsoft Research. MSR-TR-2005-72. 2005. Available at: http://research.microsoft.com/apps/ pubs/default.aspx?id=70182, accessed 20.05.2016.
7. Kaur S., Kaur H. Client Honeypot Based Malware Program Detection Embedded Into Web Pages. *International Journal of Engineering Research and Applications*, 2013, vol. 3, iss. 6, pp. 849–854. Available at: http://www.ijera.com/papers/Vol3_issue6/EM36849854.pdf, accessed 20.05.2016.
8. Zeng Y.G., Coffey D., Viega J. How Vulnerable Are Unprotected Machines on the Internet? In: *Passive and Active Measurement*. Springer, 2014, pp. 224–234 (ser. Lecture Notes in Computer Science; vol. 8362). DOI: 10.1007/978-3-319-04918-2_22
9. Maio G.D., Kapravelos A., Shoshitaishvili Y., Kruegel Ch., Vigna G. PExy: The Other Side of Exploit Kits. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 132–151 (ser. Lecture Notes in Computer Science; vol. 8550). DOI: 10.1007/978-3-319-08509-8_8
10. Rocha L. *Neutrino Exploit Kit Analysis and Threat Indicators*. SANS Institute. InfoSec Reading Room. Available at: https://www.sans.org/reading-room/whitepapers/detection/neutrino-exploit-kit-analysis-threat-indicators-36892, accessed 25.05.2016.
11. Kirat D., Vigna G., Kruegel C. Barecloud: Bare-metal Analysis-based Evasive Malware Detection. *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association. San Diego, CA, 2014, pp. 287–301. Available at: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kirat, accessed 27.05.2016.