User Intention Based Anomaly Detection

Danfeng (Daphne) Yao
Assistant Professor

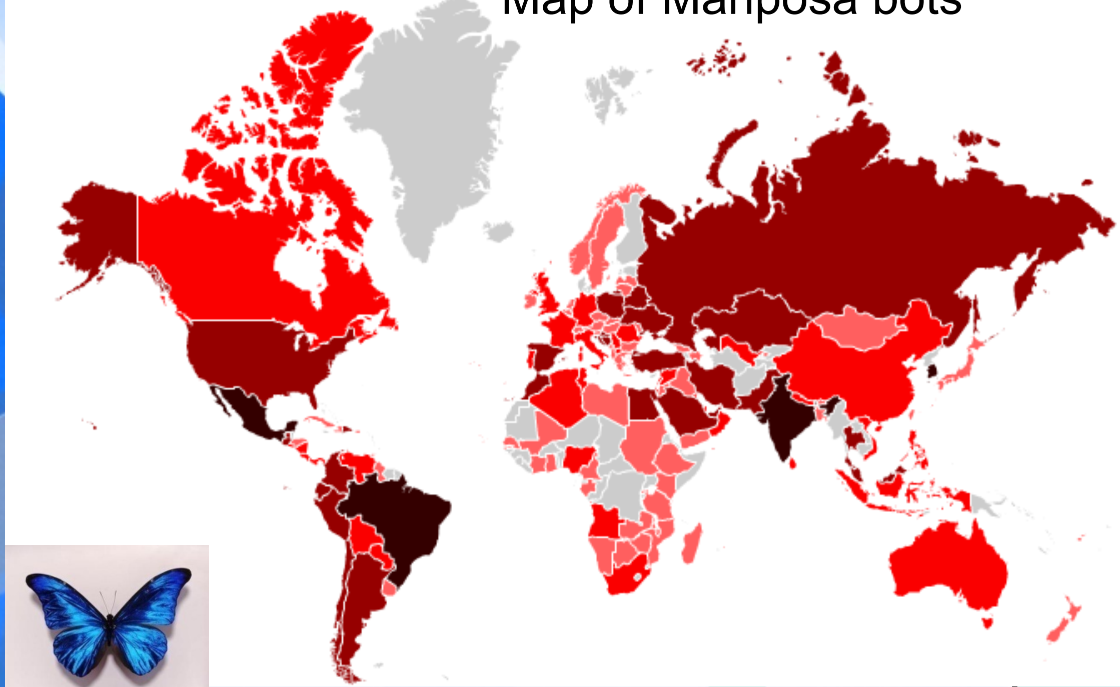
Department of Computer Science
Virginia Tech

Virginia Tech.

Verisign July 2012

Botnet threats are pervasive

Map of Mariposa bots



Mariposa botnet 12 million IPs;
Stolen data belonging to 800K users;
Malware changes every 48 hours;
Attacker uses real name in DNS

llion bots found by AT&T,
ted by botnets [GTISC 08]

Corporate

Financial loses
IP theft

nes
fo

Individual

Identity theft
Financial loses

Malware installation

E.g., **drive-by downloads**: 450,000 out of 4.5 millions URLs [Google 08]

Source: GTISC, PandaSolution

Evolving landscape of attacks

[1980's - early 1990's]
Curiosity fueled hacking: capability demonstration of hackers

[late 1990's – early 2000]
Financial driven attacks: spam, stealing credit cards, phishing, large-scale botnets

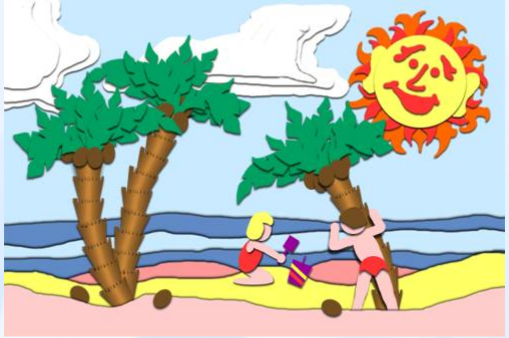
[late 2000 – present]
Targeted attacks: stealing proprietary information, information warfare

Challenges caused by:

Scale, complexity, anonymity

Internet was a friendly place. Security problem then was a day at the beach.

-- Barbara Fraser '08



Detecting malware – code vs. behavior

First academic use of term *virus* by Fred Cohen in 1984

Signature based scanning

- Analyze malware samples, extract signatures, and statically scan the file system for malicious code

But malware may encrypt/obfuscate itself

- To detect malware behaviors at run time (dynamically)
- E.g., system call execution, memory/stack access

But what about zero-day malware/exploit?

- Anomaly detection

But how to define the normalcy of a program?

D. Denning '87: anomaly detection

Problem: how to ensure system integrity?

Challenges in Winning Bot Wars

Our approach: bot detection by enforcing normal system and network patterns

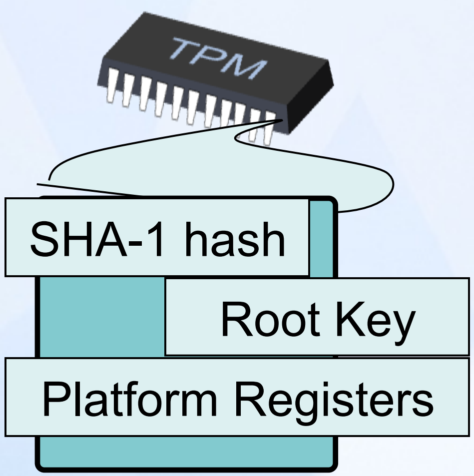
Motivation: Humans and bots have distinct patterns when interacting with computers

Challenge 1: How to find robust features?

→ User inputs and activities

Challenge 2: How to prevent bot forgery?

→ Trusted computing platform



Client-side solutions

Using our user-intention based anomaly detection techniques, a PC owner wants to know:

- What/who downloads files on the computer
- Where the keystroke is from
- Where the packet is from
- What/who causes outbound traffic
- Whether or not the apps behave

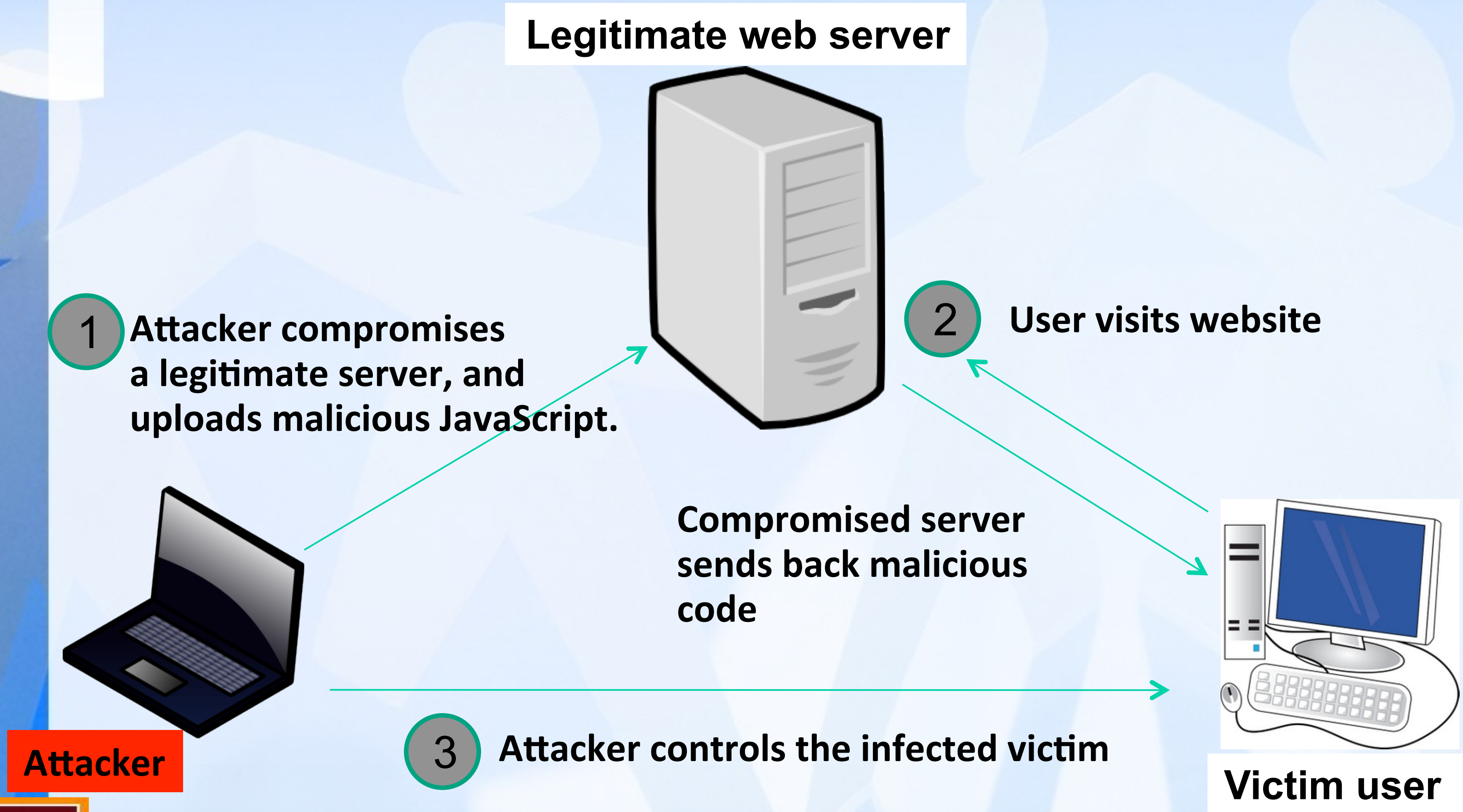
→ **For preserving system integrity**

Virginia Tech.

Know what/who downloads files on your computer

Drive-by Download Attacks

Steps of malicious code injection & host infection

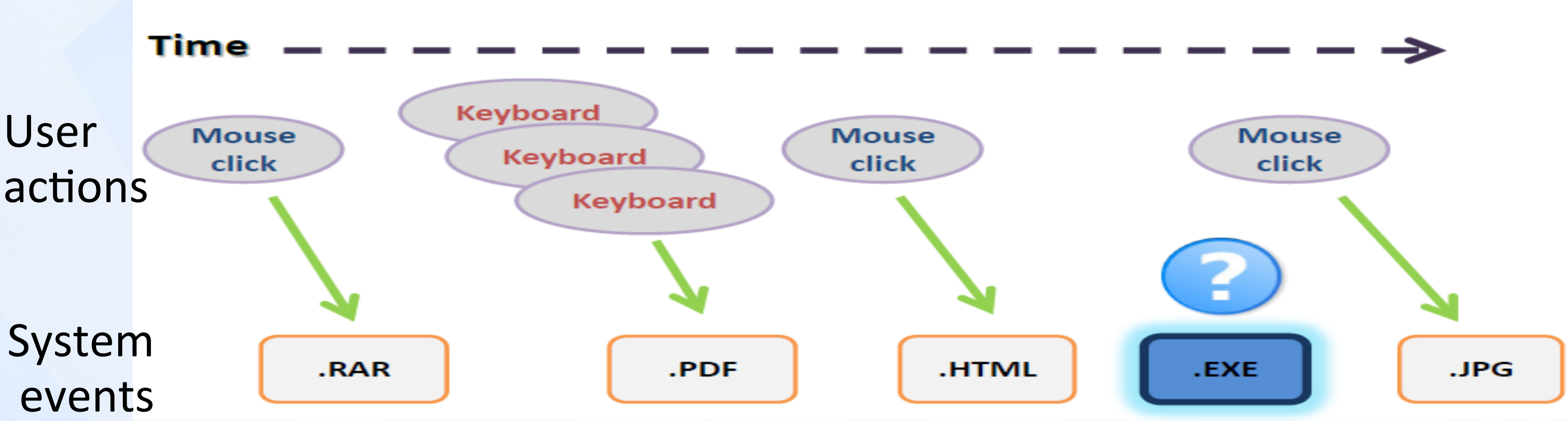


Our User Intention Based DBD Detection

Key Observation:
Legitimate system events are mostly triggered by users' actions.

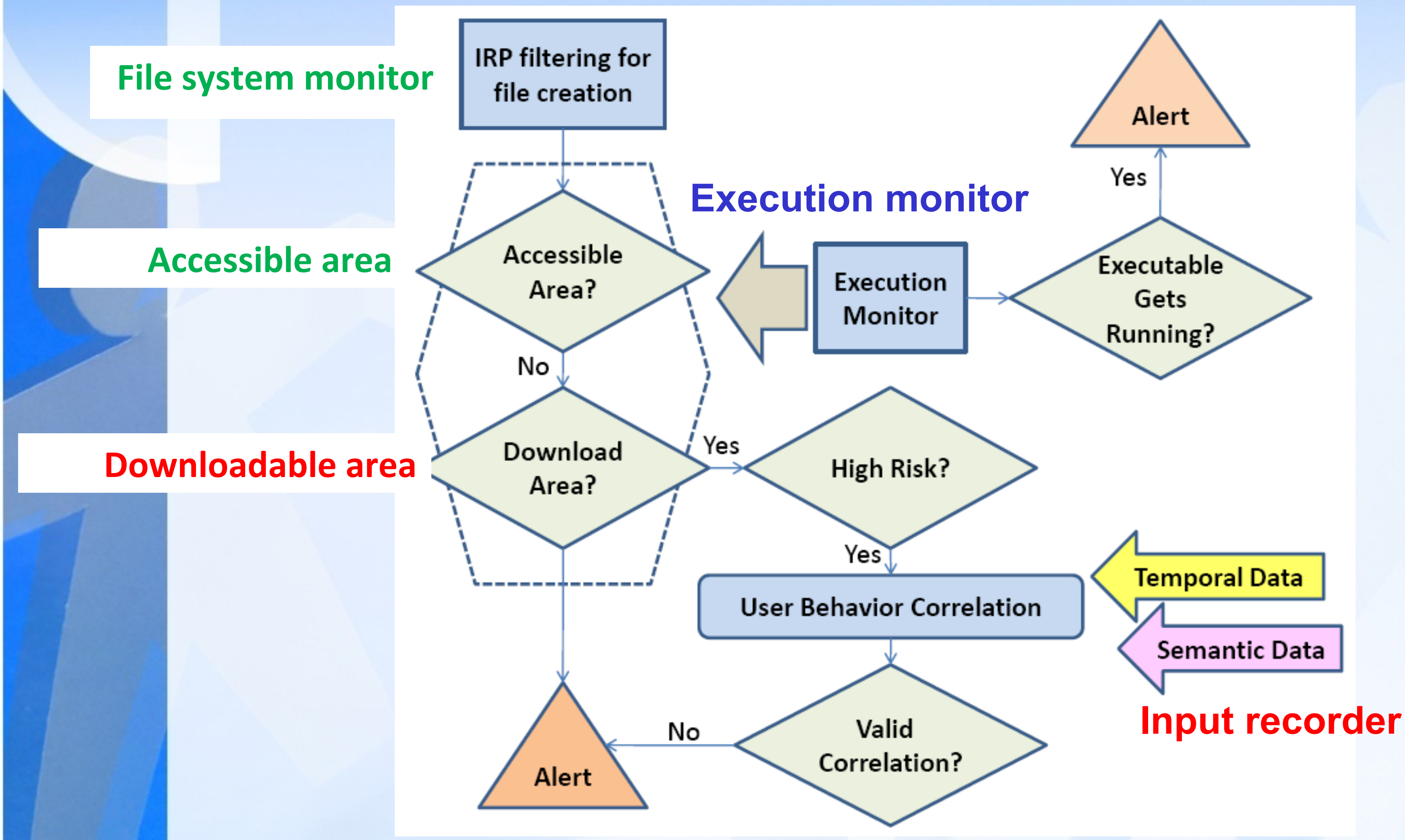
Our approach for DBD detection:

- Monitor file-creation events and user actions
- Identify the dependency between them



Challenge: Browser automatically creates files
E.g., a user indirectly triggers 482 file creation in ***Temporary Internet Files*** folder and 47 in ***Cookies*** directory within 30 minutes of surfing.

Components and work flow



DeWare prototype in Windows 7 Ultimate edition

Dependency Rules Among Events

A file creation event and its triggering user event need to satisfy dependency rules

- ***Rule 1* File properties of events match.**
  - The file user confirms to create should be same as the one actually created.
- ***Rule 2* URLs match.**
  - The file should be downloaded from the URL that user requests.
- **Rule 3 Process properties of events match.**
  - The process that receives input should be the one creating the file.
- **Rule 4 Temporal constraint is satisfied.**
  - A legitimate file creation event should take place within a short threshold after a valid user-input event.

Virginia Tech. VT

Evaluation of DBD Detection Ability

Against popular DBD exploits:

– **We successfully detected the lab reproduced exploits:**

- *Heap Feng Shui attack* 風水
- *HTML Object Memory Corruption Vunerability*
- *Superbuddy through AOL activeX control*
- *Adobe Flash player remote-code execution*
- *Microsoft Data Access Component API misuse*
- *DBD exploiting IE 7 XML library*

Note: DeWare is not for detecting code injection attack -- code or DLL injected into the memory of a legitimate process

Evaluation of DBD Detection Ability

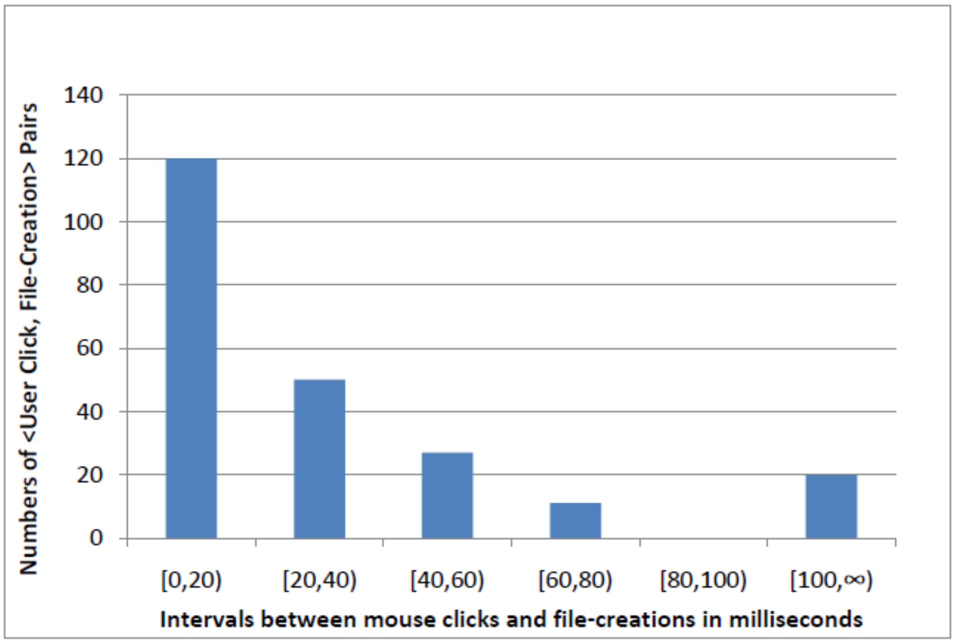
Against real-world malicious websites listed by

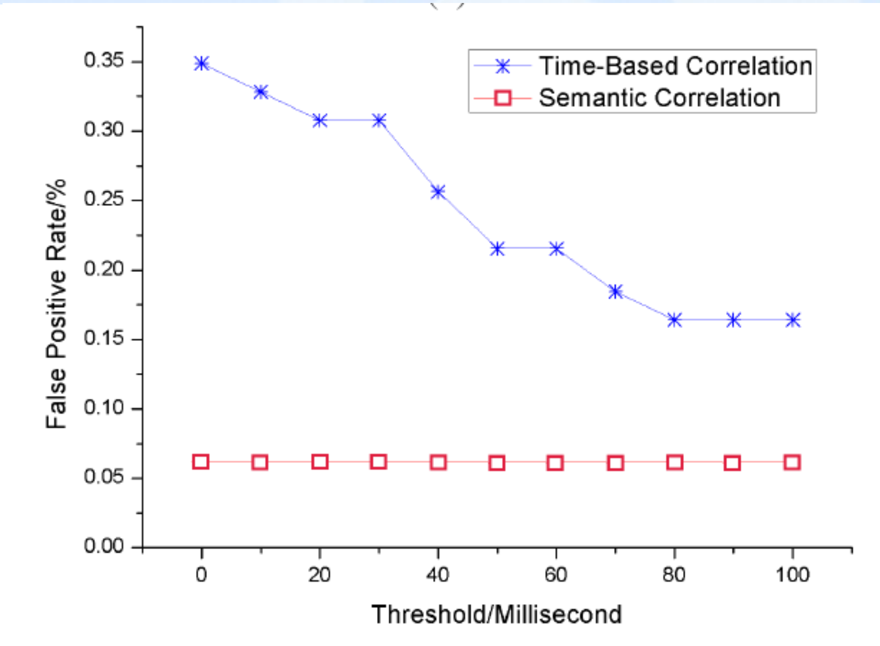
www.malwaredomainlist.com
www.malwareurl.com

- 84 out of 142 malicious websites were detected by DeWare
  - Some websites track incoming requests and a second visit would not trigger exploit
- Malicious websites download .exe and/or .dll files
  - E.g., to \Temp folder
- Popular exploit kits are used:
  - Phoenix exploit kit
  - Eleonore exploits pack
  - Targeting at multiple vulnerabilities including Flash, PDF, Java, and browser

Two False Positive Evaluations

- We automatically evaluated top 2000 websites from Alexa
- DeWare triggered zero false alarm





- Another false positive evaluation based on 21 user study data
- The number of false alarms is small, less than 1%

Evaluations on commodity software on IE 7 XML DBD attacks

McAfee, Kaspersky, AVG

Security Software			Reaction
Product	**Driver Engine Version**	**Definition**	
Trend Micro Internet Security Pro	v8.952	Pattern version 6.289	No detection
		Pattern version 6.587.50	No detection
Microsoft Security Essentials		Virus Definition 1.69.825 Spyware Definition 1.69.825	Detected. User clicked clean the threats, but DBD files were still downloaded and not deleted by MSE.
360 Safeguard	v3.0	.1112	No detection
	360 v6.0.1	2008-6-16	No detection
	360 v6.0.1	2008-10-27	No detection
	360 v6.0.2	2009-10-14	Detected Heap Spray attack, shutdown iexplorer.exe
Zonealarm Pro	7.0.483	Anti-spyware engine 5.0.189	Captured a.exe trying to access internet. Clicked "Deny", but H.exe was still downloaded successfully
	8.0.400	Anti-spyware engine 5.0.209	
	9.1.008	Anti-spyware engine 9.1.008	

Drive-by download detection work appeared in:

Kui Xu, Danfeng Yao, Qiang Ma, and Alexander Crowell. Detecting Infection Onset With Behavior-Based Policies. In ***Proceedings of the Fifth International Conference on Network and System Security (NSS)***. Milan, Italy. Sep. 2011.

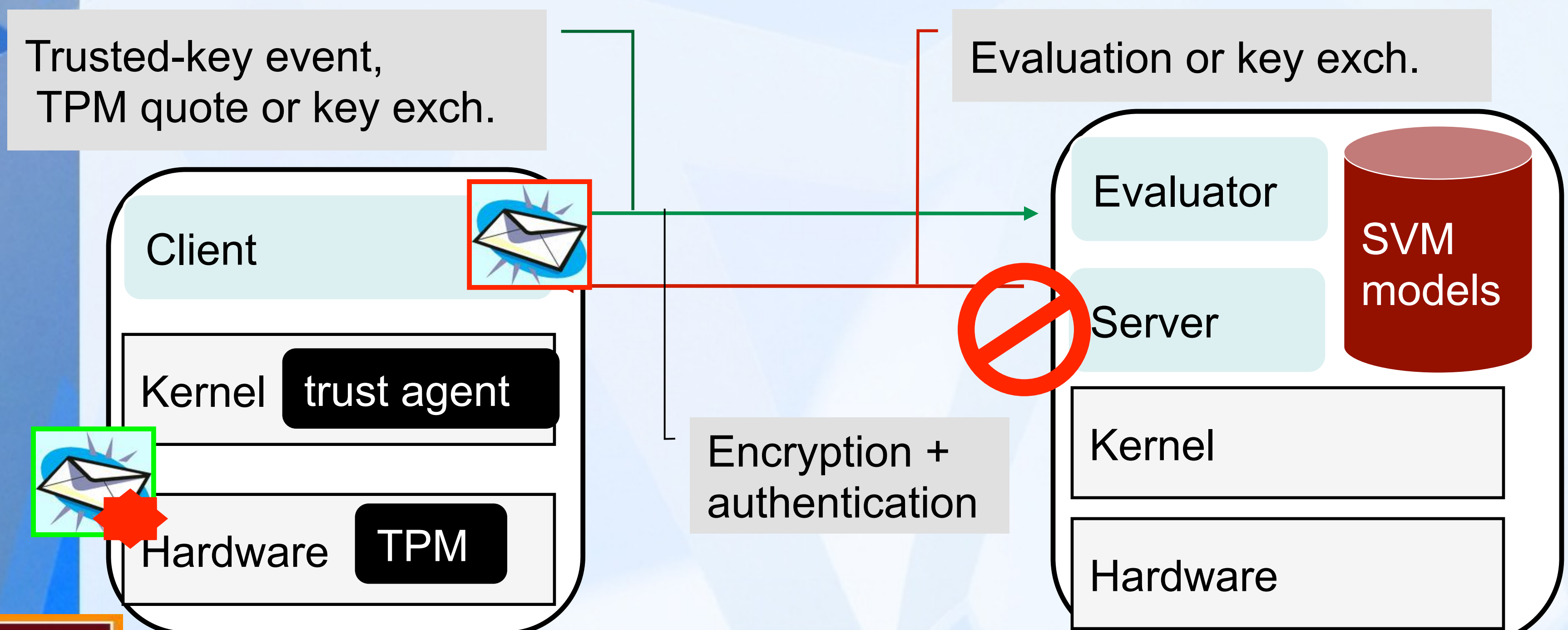
Know where your keystroke is from

Preventing Strong Adversaries With TPM

A stronger adversary may:
- Gain root on the computer
- Collect the owner's keystroke information
- Tampering TUBA client

Our prototype on Intel Core 2 Duo (INT-C0-102) following TPM Interface Spec 1.2

Our goal: to prevent fake key event injections & tampering TUBA

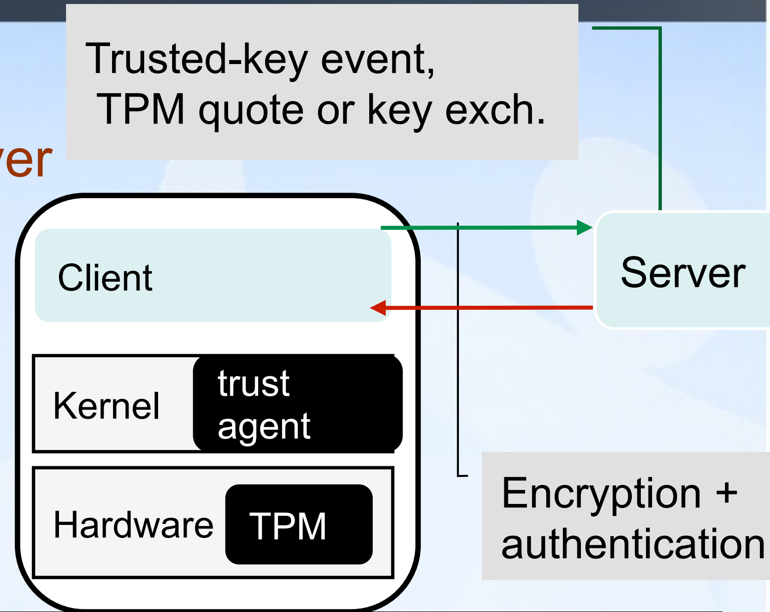


Related TCB/TPM work [MPR NDSS 09] [GBMR, NSDI 08] [MPPRI, EuroSys 08]

Highlights in TUBA Integrity Service

1. Server verifies trusted boot of client
2. Key exchange between agent & server
3. Trust agent signs keystroke events
4. Client relays signed events

Secrecy of Signing key is guaranteed



Sign a packet (SHA1) with a 256-bit key: 18.0 usec

Encrypt a packet (AES-CBC) with a 256-bit key: 67.6 usec

(Averaged on 1312 keystroke events with TPM key initiation.)

Bandwidth (i.e., communication overhead): 13 KBps

Summary: Robust TUBA introduces minimal overhead and practically causes no delay even for a fast typist

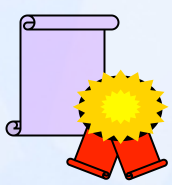
Our Approach: Cryptographic Provenance Verification (CPV)

Data-provenance integrity – origin of kernel-level data not spoofed

CPV - a robust attestation mechanism that ensures true origin of data

TUBA embodies our CPV approach

CPV differs from traditional digital signatures



Signs a document

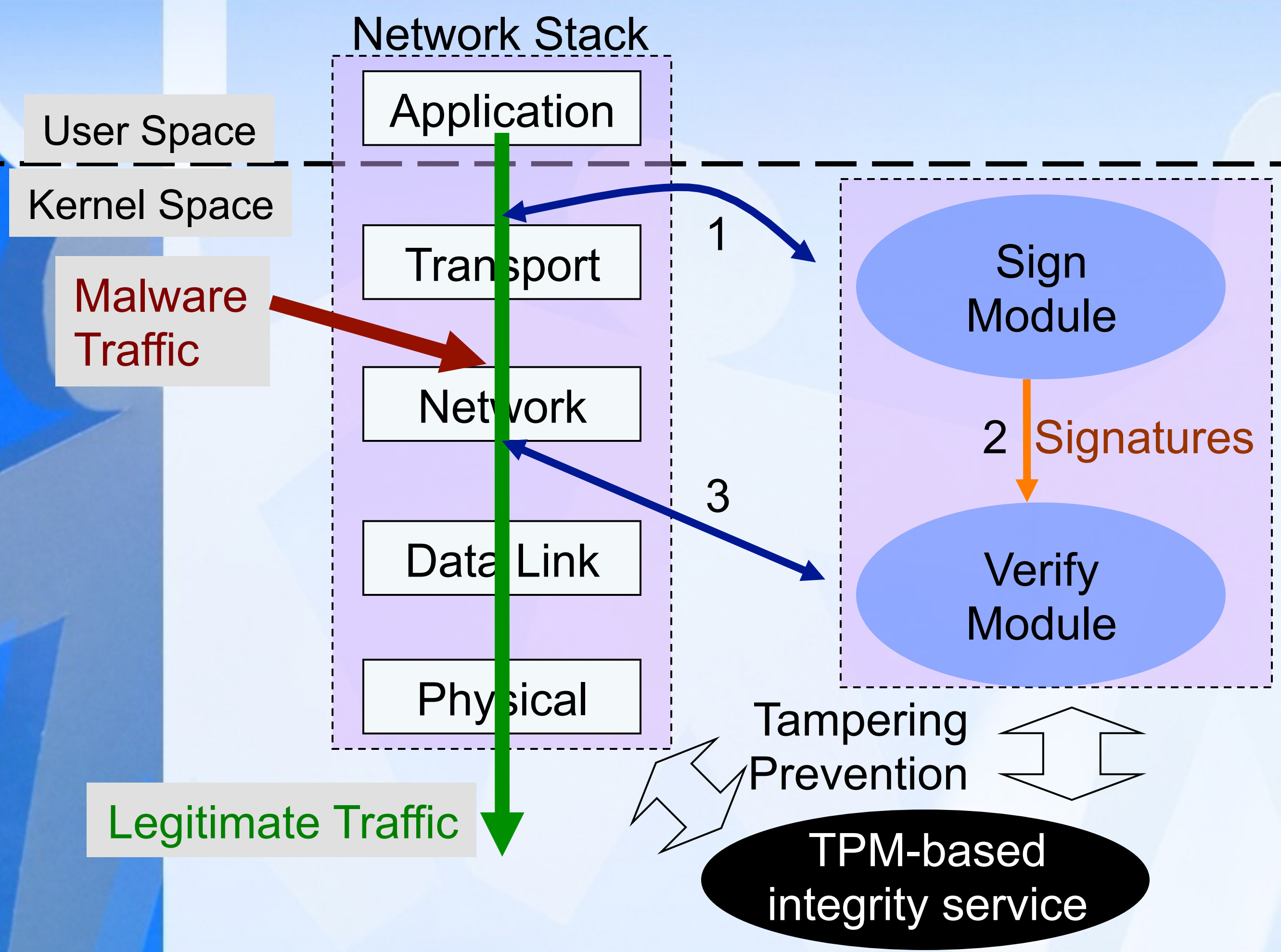
Signer knows what to sign and what not

Virginia Tech

Know where your outbound network packet is from

i.e., to catch all outbound traffic from a host for inspection

Apply Cryptographic Provenance Verification to Network Stack



2 Kernel Modules:
- Sign Module
- Verify Module

Key management:
- Key derived from TPM

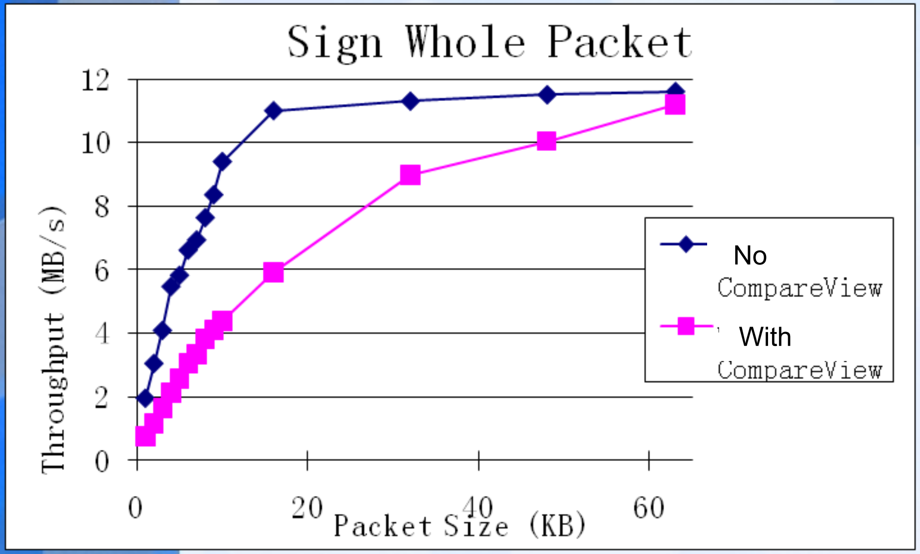
Technicalities:
- Defragmentation
- Signature transfer

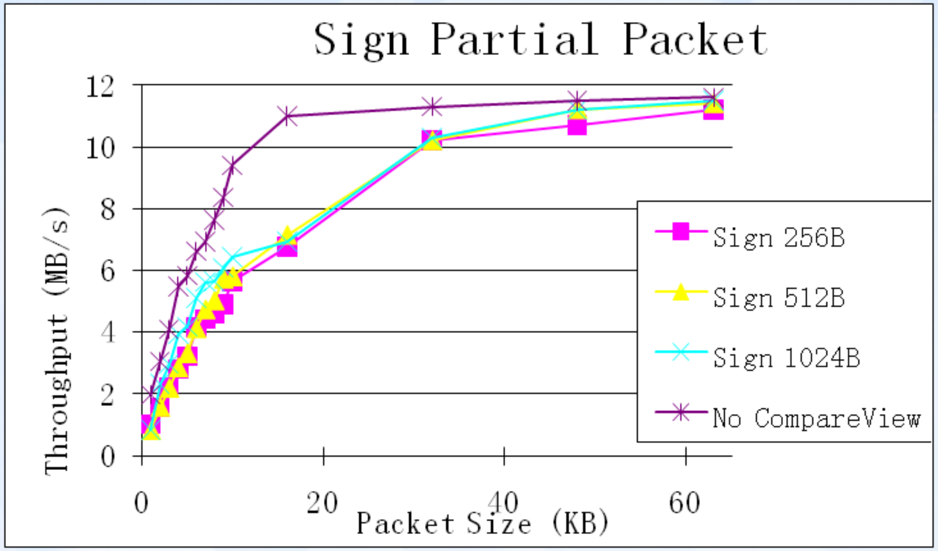
Prototype in Windows

Our solution enables advanced traffic inspection – no packet left behind

Throughput Analysis in CompareView

- As packet size increases, overhead decreases
- < 5% overhead for 64KB packet size
- Signing partial packet reduces overhead





Successfully detected several real-world and synthetic rootkit-based malware

Fu_Rootkit, hxdef, AFXRootkit, our proof-of-concept rootkit

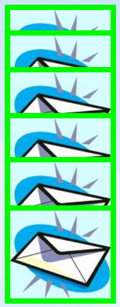
Summary: Our work enables robust personal firewall

Cryptographic provenance verification work appeared in:

Kui Xu, Huijun Xiong, Chehai Wu, Deian Stefan, and Danfeng Yao. **Data-Provenance Verification For Secure Hosts.** ***IEEE Transactions of Dependable and Secure Computing (TDSC).*** 9(2), 173-183. March/April 2012.

Know what/who causes your outbound traffic

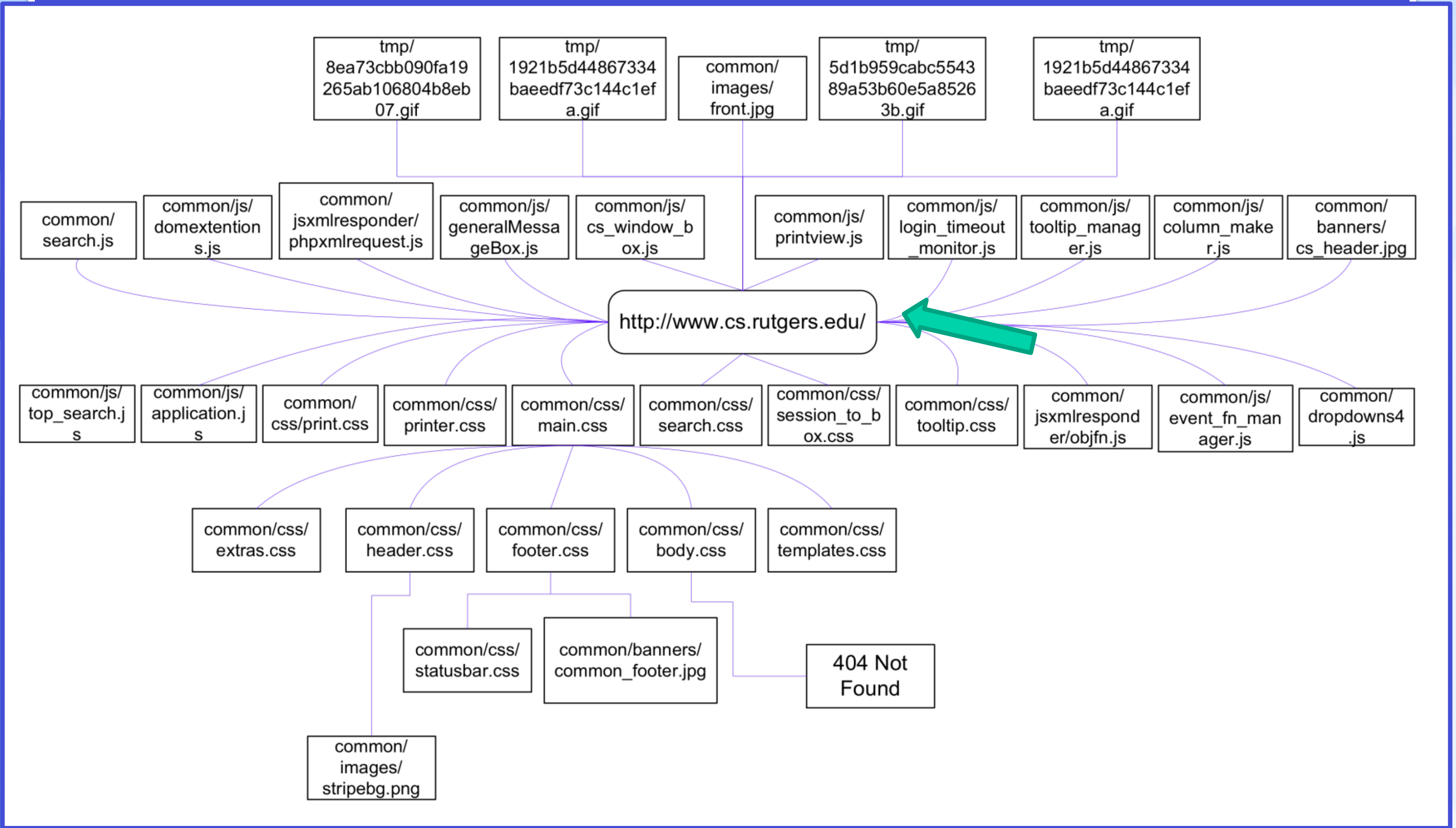
Cause and Effect in Traffic Anomaly Detection



How to distinguish the malicious outbound packets from the legitimate ones on a host?

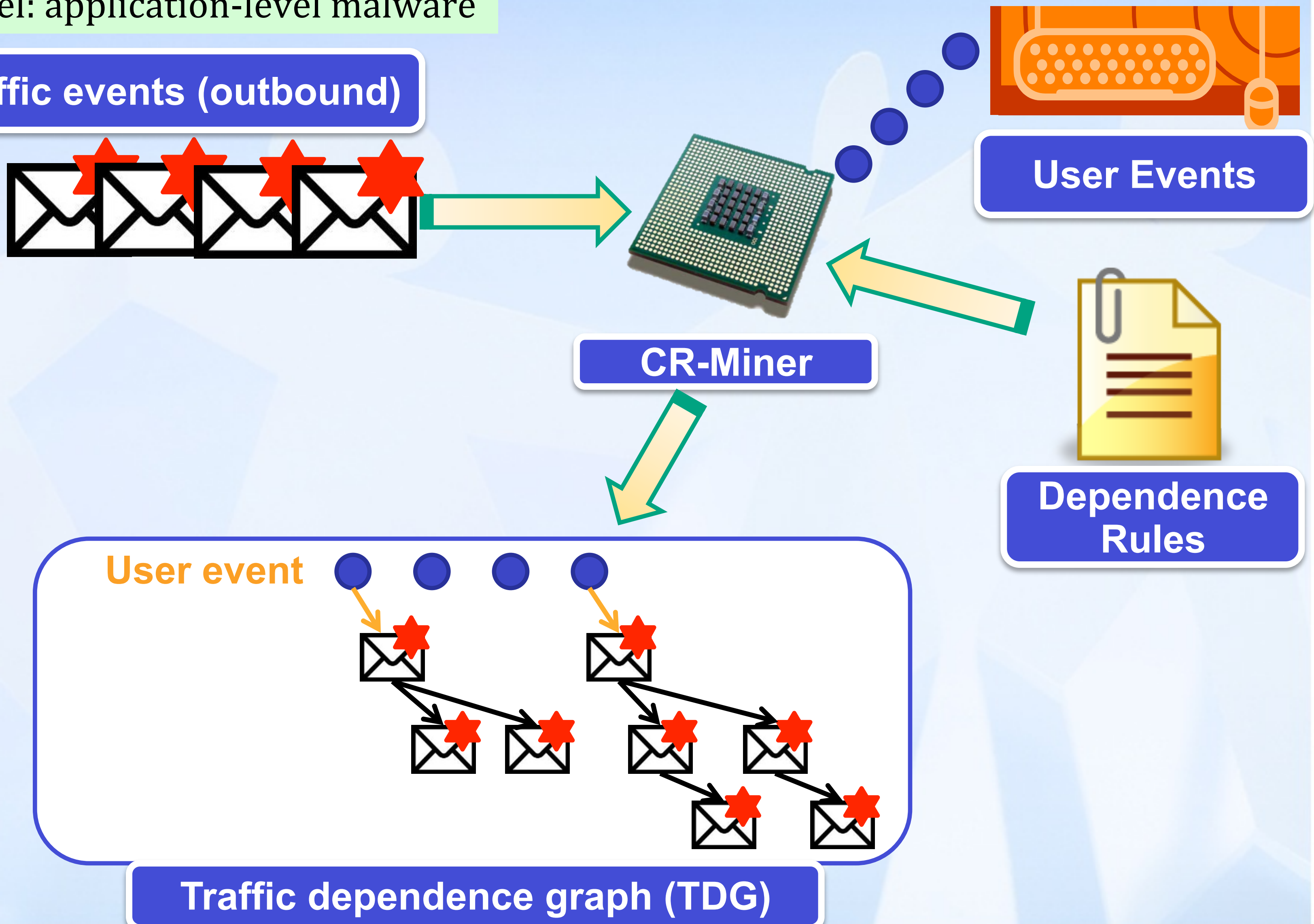
Our approach: To enforce ***dependence*** among outbound traffic

A Technical Challenge



Work Flow of CR-Miner

Threat model: application-level malware



Events and their attributes

Dependence rules specify relations of attributes of dependent events

User events

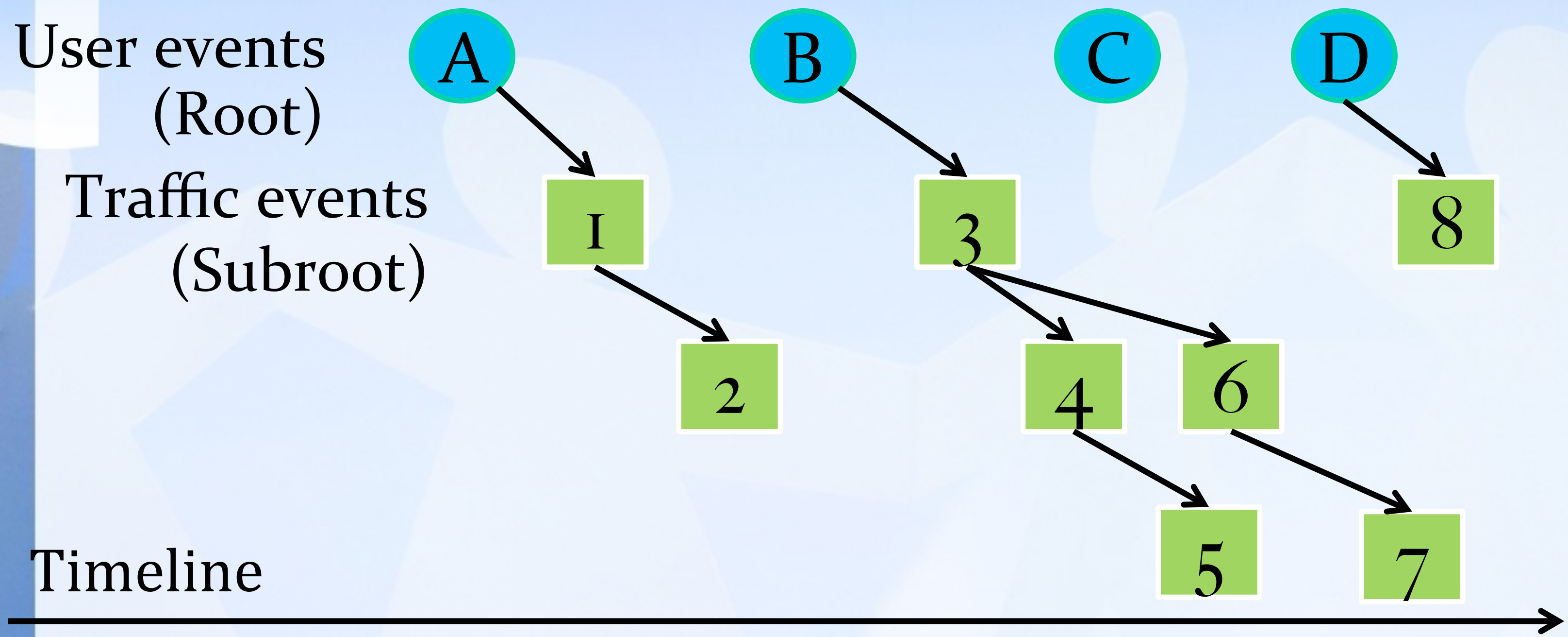
	Timestamp	Event Name	Value	URL
A	0:0:01.077	KeyDown	Return	http://www.engadget.com/
B	0:0:02.910	MouseClick - Left	X=1069 Y=474	http://www.cnet.com/
C	0:0:03.000	Wheel	-120	N/A

Traffic events

	Timestamp	Object Requested	Remote Domain Name	Referrer
1	0:0:02.863	/	www.engadget.com	http://www.engadget.com/
2	0:0:02.873	/media/main.css	www.engadget.com	http://www.engadget.com/...
3	0:0:03.113	/	www.cnet.com	null

Virginia Tech.

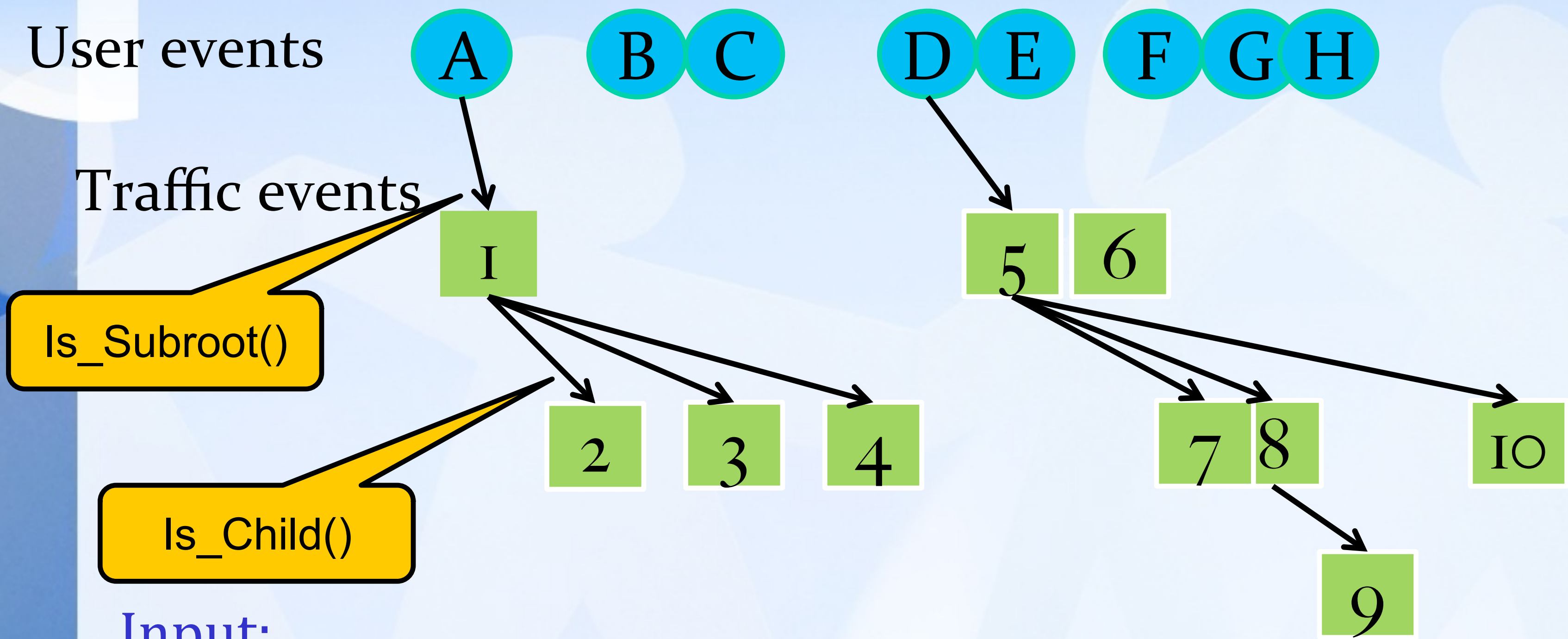
Definitions in Our Traffic Dependency Graph (TDG)



Definition of security: a legitimate traffic event should belong to a tree in a TDG that is rooted at a legitimate user event.

Otherwise, it is a vagabond traffic event

Our BFS-Based Algorithm to Construct Traffic Dependence Graph



Input:

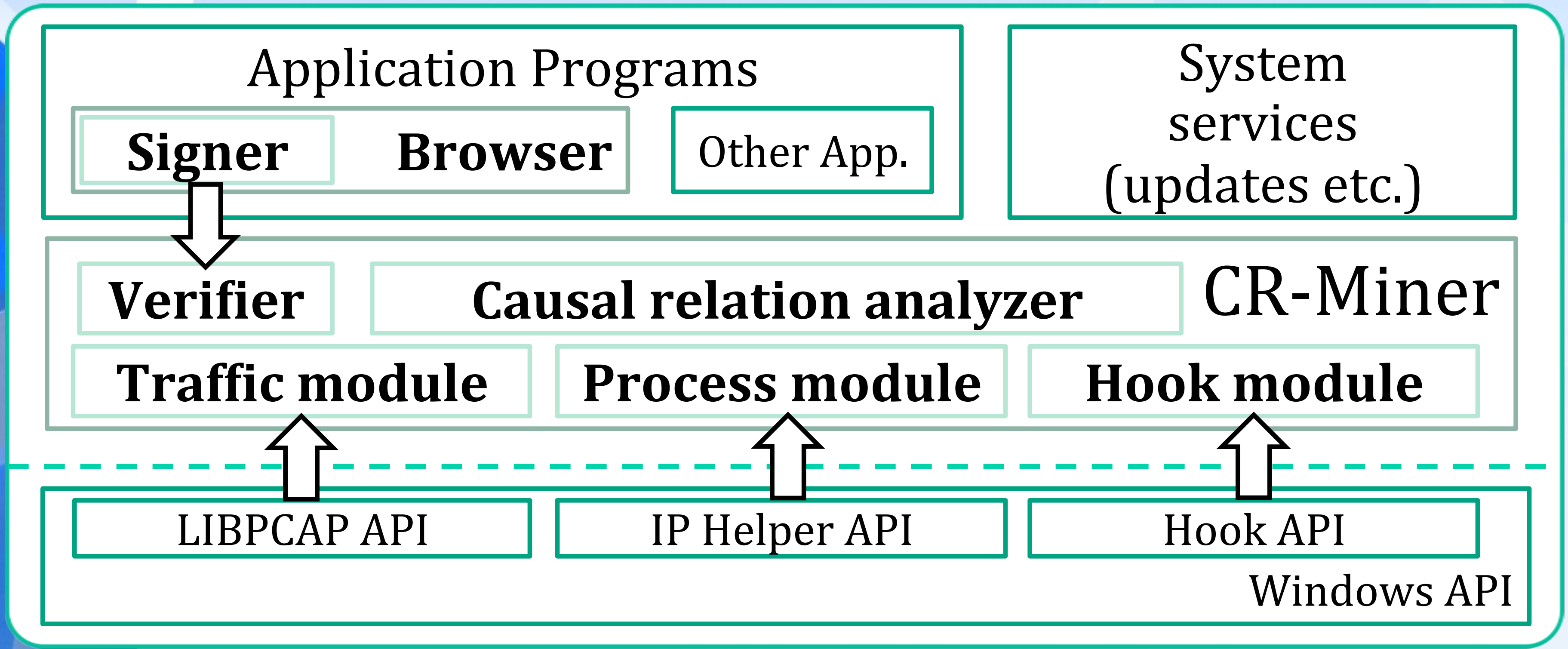
- an existing TDG (trees of events, which root at user events)
- a new outbound traffic event ***q***

Output:

- whether or not ***q*** is legitimate

Implementation Architecture

Our prototype in Windows is called CR-Miner.



Signer and verifier for the integrity of HTTP requests with MAC

Highlights on Experiments

User study with 20 participants; 30-minute surfing for each user

Hit rate: percentage of traffic events whose parents are identified by CR-Miner

1. How accurate is the dependency inference algorithm?
   - ≥ 98% hit rates for all users
   - Average 99.6% with white listing (0.4% contains true positives)
   - 99.72% for top 20 Alexa.com websites (i.e., 0.28% false positives)
2. Does the inference accuracy suffer in noisy traffic?
   - 99.2% accuracy in two-user merged data set
3. Can we detect real-world stealthy malware traffic?
   - Infostealer spyware
   - Proof-of-concept password sniffier (malicious Firefox extension similar to Firespyfox)

Virginia Tech

Traffic dependency work appeared in:

Hao Zhang, Danfeng Yao, Naren Ramakrishnan, and Matthew Banick. User Intention-Based Traffic Dependence Analysis for Anomaly Detection. ***Workshop on Semantics and Security (WSCS***), in conjunction with the *IEEE Symposium on Security and Privacy*. San Francisco, CA. May 2012

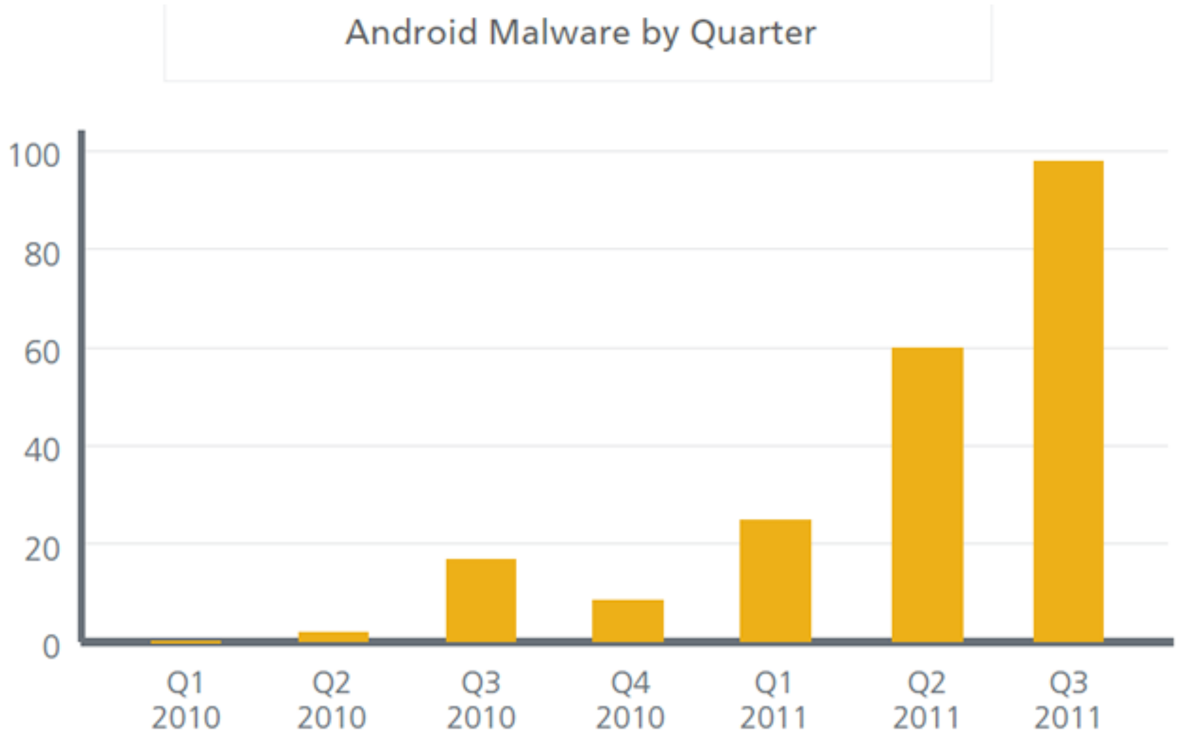
An earlier related work of ours appeared in:

Huijun Xiong, Prateek Malhotra, Deian Stefan, Chehai Wu, and Danfeng Yao. User-Assisted Host-Based Detection of Outbound Malware Traffic. In ***International Conference on Information and Communications Security (ICICS)***. Beijing, China. Dec. 2009

Know whether or not your apps behave

A white-box approach

Legitimate or Malicious: an app-classification problem

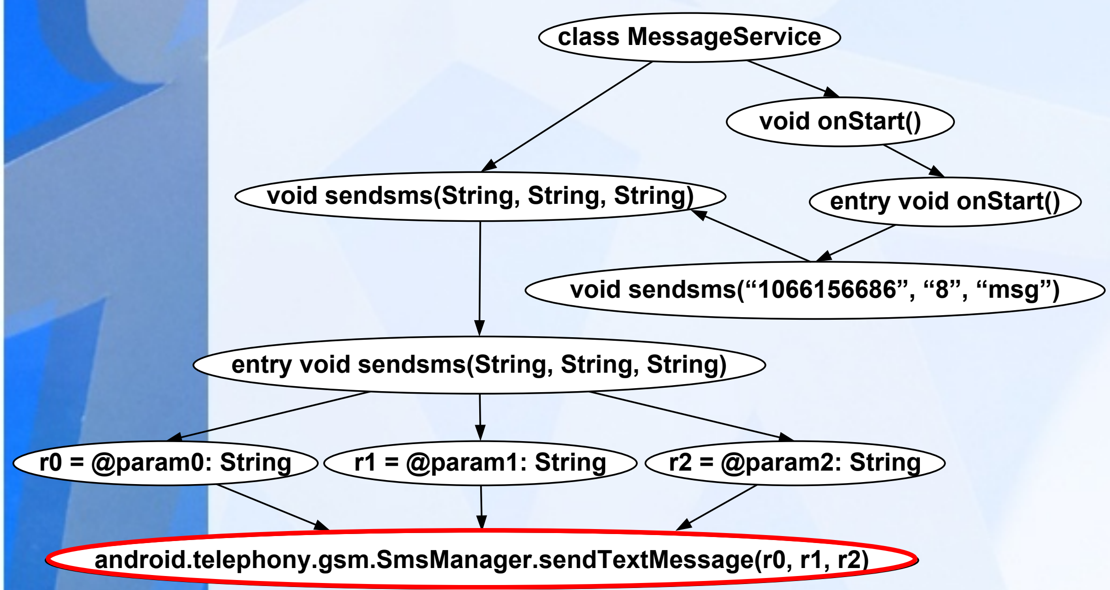


Problem: How to classify unknown apps as benign or malicious?

Source: http://news.cnet.com/8301-1009_3-57328575-83/androids-a-malware-magnet-saysmcafee/?tag=mncol;topStories

Example of Malicious App: HippoSMS

This malware sends SMS messages to a hard-coded premium-rated number without the user's awareness



A Data Dependence Graph

```
public class MessageService{
   .....
     public void onStart(){
        sendsms("1066156686", "8", "");
     }
     public void sendsms(param1,
param2, param3){
         .....

localSmsManager.sendTextMessage(
         param1, param2, param3);
     }
}
```

Malicious code

What is the norm? How to enforce it?

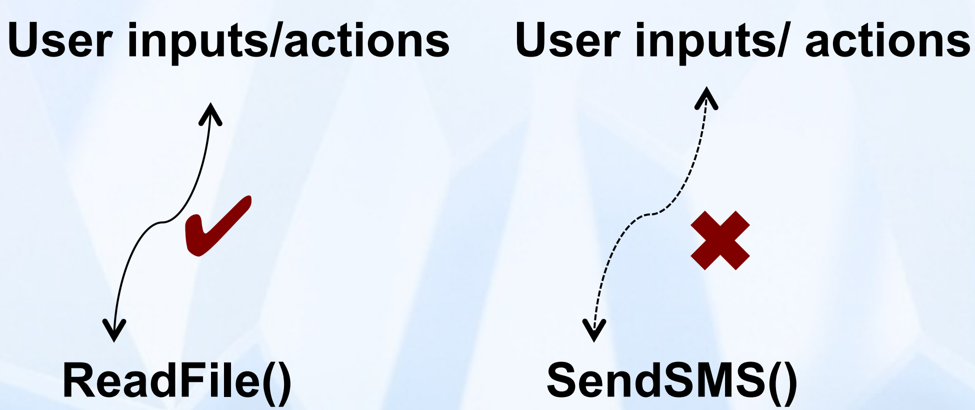
Requests to access system resources
are usually based on user inputs / actions in apps

Our approach:

Identify dependences between function calls and user inputs through program analysis

Resources to protect from malicious programs:

- File system access
- Network access
- Sensitive/personal data



Our User-Centric Dependence Based Anomaly Detection Approach

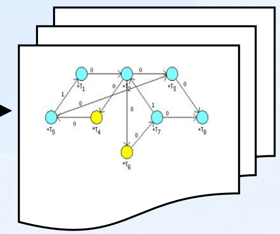
Check dependency rule

Program source / bytecode

Identify user inputs/actions

Identify critical functions API

Data-flow analysis



Data Dependence Graph

Identify sensitive paths

Classification results

Our Static Analysis Tool:

We utilize definition-use structures provided by Soot (a static analysis toolkit for Java)

Our tool can analyze Java bytecode / source code

Virginia Tech

Soot Bytecode Optimization Framework

Java

Work in progress

Evaluation Results on Legitimate and Malicious Android Apps

Most malware apps tested do not satisfy data dependence requirement

	App/Malware Name	# of User Inputs/ Actions (Source)	% of Sensitive Func. Calls without User Inputs	Library of Sensitive Function Calls
Legitimate	SendSMS	3	0%	android.telephony.gsm
	BMI Calculator	2	0%	android.app.Activity
	BluetoothChat	2	0%	java.io.OutputStream
	SendMail	4	0%	android.app.Activity
	Tip Calculator	4	0%	android.widget
Malicious	GGTracker.A	0	100%	org.apache.http.impl.client
	HippoSMS	0	100%	android.telephony.gsm android.content.ContentResolver
	Fakeneflic	3	0%	org.apache.http.impl.client
	GoldDream	0	100%	android.content.Context java.io.FileOutputStream
	Walk & Text	0	100%	android.content.ContentResolver org.apache.http.impl.client
	RogueSPPush	0	100%	android.telephony.gsm android.content.ContentResolver
	Dog Wars	0	100%	android.telephony.gsm android.content.ContentResolver

Security Analysis

Attacks	Countermeasures
Phishing apps / social engineering apps	Site authentication and user education
Using superfluous user inputs and actions	Easy to detect by using our approach to track the dependency
Code obfuscation or Java reflection	Dynamic taint analysis

Program analysis work appeared in:

Karim Elish, Danfeng Yao and Barbara Ryder.
User-Centric Dependence Analysis For Identifying Malicious Mobile Apps.
In ***Proceedings of the Workshop on Mobile Security Technologies (MoST)***, in conjunction with the IEEE Symposium on Security and Privacy.
San Francisco, CA. May 2012.

Future Work on User-Intention Based Anomaly Detection

User-intention based anomaly detection is a promising approach; we've demonstrated its use in detecting anomalies in

- network traffic,
- file system events,
- apps,
- keystrokes …

Future work:

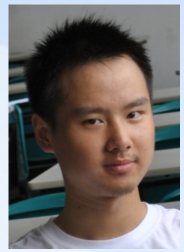
- More investigation on white box anomaly detection and analysis
- Android based mobile system integrity
-

Personnel in Yao group

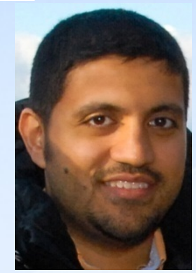
Current Ph.D. students


Kui Xu


Huijun Xiong


Johnny Shu


Tony Zhang

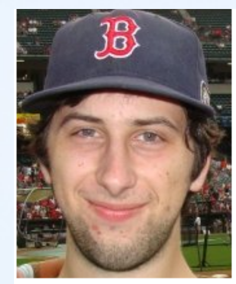

Hussain Almohri


Karim Elish

Previous group members


Deian Stefan (REU 08)


Chehai Wu (MS 09)


Matt Banick (BS 11)

Funding Sources:

- NSF CAREER, ARO, S2ERC (Verisign)