Cross-Tenant Side-Channel Attacks in PaaS Clouds

Y. Zhang, A. Juels, M. K. Reiter, T. Ristenpart
Presenter: Serif Yesil

September 29, 2017

Attacks in PaaS Clouds

A framework for cache based side channel attacks for the attacks between tenants on a commercial PaaS clouds

Attacks in PaaS Clouds

"Platform as a service (PaaS) or application platform as a service (aPaaS) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app." [1]

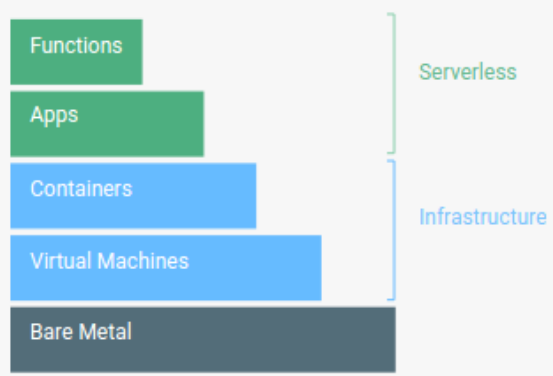

Figure 1: PaaS structure[2]

[1] https://en.wikipedia.org/wiki/Platform_as_a_service
[2] https://cloud.google.com/appengine/

Challenges

- Implementing side channel attacks in PaaS setting is not straightforward
- Identifying suitable targets to attack in PaaS environment is a problem

Contributions

- Automated cache based side channel attack framework based on nondeterministic finite automaton
  - Number of times a certain execution path is visited
  - Precise time a specific code piece executed
  - Direction taken in a specific branch

Contributions

- Automated cache based side channel attack framework based on nondeterministic finite automaton
  - Number of times a certain execution path is visited
  - Precise time a specific code piece executed
  - Direction taken in a specific branch
- Automated scheme to confirm co-location with the victim in a commercial cloud service

Contributions

- Automated cache based side channel attack framework based on nondeterministic finite automaton
  - Number of times a certain execution path is visited
  - Precise time a specific code piece executed
  - Direction taken in a specific branch
- Automated scheme to confirm co-location with the victim in a commercial cloud service
- Approach is tested with 3 use cases
  - Inferring sensitive user data
  - Password-reset attacks
  - Saml-based single sign on attacks

Contributions

- Automated cache based side channel attack framework based on nondeterministic finite automaton
  - Number of times a certain execution path is visited
  - Precise time a specific code piece executed
  - Direction taken in a specific branch
- Automated scheme to confirm co-location with the victim in a commercial cloud service
- Approach is tested with 3 use cases
  - Inferring sensitive user data
  - Password-reset attacks
  - Saml-based single sign on attacks
- Deployed in real world environment
  - DotCloud and OpenShift

Outline

1. Background
2. Automated Attack Framework
3. Detection of Co-location
4. Case Studies
5. Discussion

Background

PaaS Sharing and Isolation

- PaaS cloud allows users to upload interpreted code
- Runtime environment is provided by the host
- Executes the code in provider managed operating systems
- To maximize utilization PaaS systems are multi-tenant

PaaS Sharing and Isolation

- Isolation is provided in 4 different ways
  - Runtime-based: Tenants are isolated by their application runtimes
  - User-based: Traditional user based isolation in the same host OS
  - Container-based: Isolate users based on containers. A container executes a group of processes that has distinct kernel namespaces and resource allocation quotas.
  - VM-based: Each user provided a VM as in IaaS.

PaaS Sharing and Isolation

- Isolation is provided in 4 different ways
  - **Container-based: Isolate users based on containers. A container executes a group of processes that has distinct kernel namespaces and resource allocation quotas.**

PaaS cloud	URL (http://...)	Isolation
AppFog	www.appfog.com	User
Azure	azure.microsoft.com	VM
Baidu App Engine	developer.baidu.com/en	Container
Cloud Foundry	cloudfoundry.org	User
DotCloud	www.dotcloud.com	Container
Elastic Beanstalk	aws.amazon.com/elasticbeanstalk/	VM
Engine Yard	www.engineyard.com	VM
Heroku	www.heroku.com	Container
HP Cloud Application PaaS	www.hpcloud.com/products-services/application-paas	Container
Joyent SmartOS	www.joyent.com	VM
OpenShift	www.openshift.com	Container
WSO2	wso2.com/cloud	Runtime

Figure 2: Isolation techniques

Threat Model

- Malicious customers in a container-based isolation environment
- Attacker has two main goals
  - Co-locate its malicious code/instance within the same host OS as the victim
  - Extract sensitive information from the victim

Automated Attack Framework

Attack Framework

- A nondeterministic finite automaton based automated attack framework is provided
- FLUSH-RELOAD based side channels are exploited

FLUSH-RELOAD

- FLUSH: Attacker flushes specific cache lines containing target memory regions to monitor
- FLUSH-RELOAD interval: Attacker waits for some time for victim to fill the cache
- RELOAD: Attacker reload its cache lines to the cache and monitor the access time
    - Faster access suggests corresponding lines are used by the victim

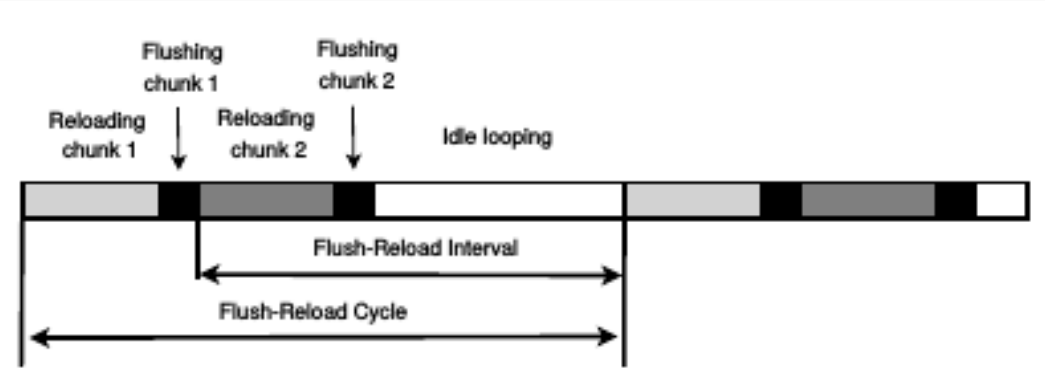

Figure 3: Flush-Reload

FLUSH-RELOAD

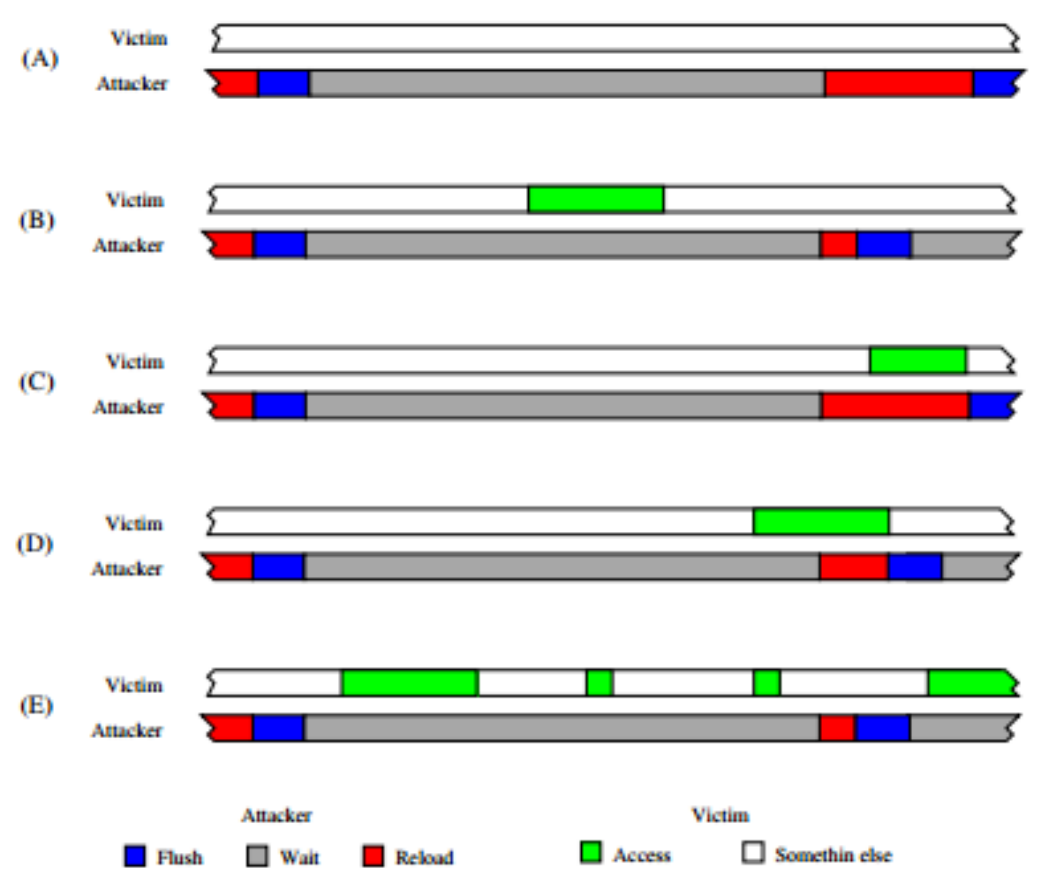

Figure 4: FLUSH-RELOAD[3]

[3]Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack

Automated Framework

- Goal: Design an attack NFA that defines the order in which different chunks (cache lines) should monitored using FLUSH-RELOAD attack

Automated Framework

- Goal: Design an attack NFA that defines the order in which different chunks (cache lines) should monitored using FLUSH-RELOAD attack
- Attacker analyzes the control-flow graph of the shared executable with the victim
  - CFG nodes are basic blocks of instructions
  - CFG edges are possible execution paths

CFGs to Attack NFAs

- Functions to define the attack environment
  - BBToChunks : $B \rightarrow 2^C$
  - NFA: $(Q, \sum, \delta, q_0, F)$
  - Q: set of states
  - $\sum$: set of symbols, $\sum = C \times \mathbb{N} \times \mathbb{N}$
  - δ: $Q \times \sum \rightarrow Q$ transition function
  - F (accepting states) is subset of Q
  - For each state q, mon(q) gives the chunks that will be monitored in the state
- After NFA is constructed, attacker can start monitoring by sending requests to victims application

Construction of Attack

- Attacker knows the shared executables
- Attacker can trigger victims execution by sending requests to victims application
- Attacker can monitor co-located victim

Construction of Attack

- Disassembling these shared executables
- Manually analyzing these partial CFGs and decide code blocks to monitor
- Constructing the attack NFA with the help of online training
  - Adversary monitors all chunks of interest at once and triggers the victims activity that he would like to capture
  - Adversary finds optimal timing parameters

Example: Password-Reset Attack

- Aims to compromise pseudorandom number generators (PRNGs) that are used for password reset requests
- The target is PHP runtime and system time functions
- Detect the system calls and replay the internal state of PRNG to reproduce the password reset URL

Example: Password-Reset Attack

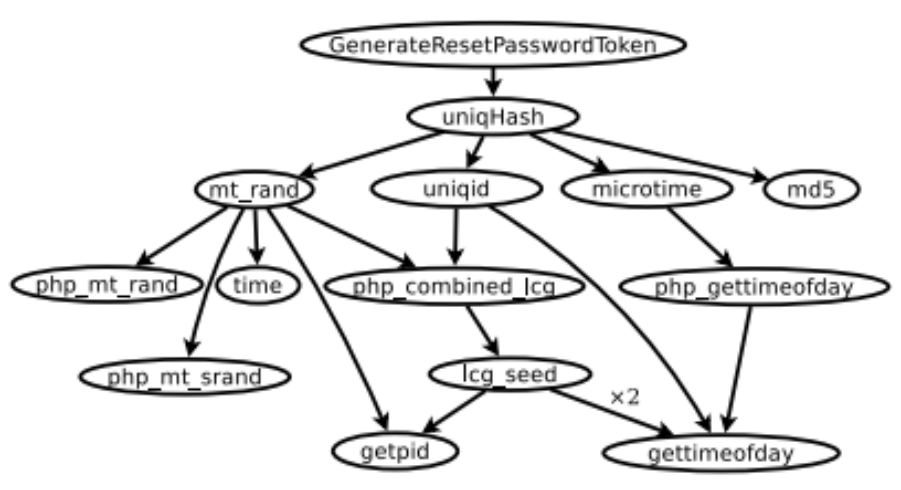

Figure 5: CFG for password reset

- Only sources of entropy **gettimeofday()**, **getpid()**, and **time()**
- **gettimeofday()**, **time()** can be called right after attacker detects their execution.
- Only unknown is **getpid()**

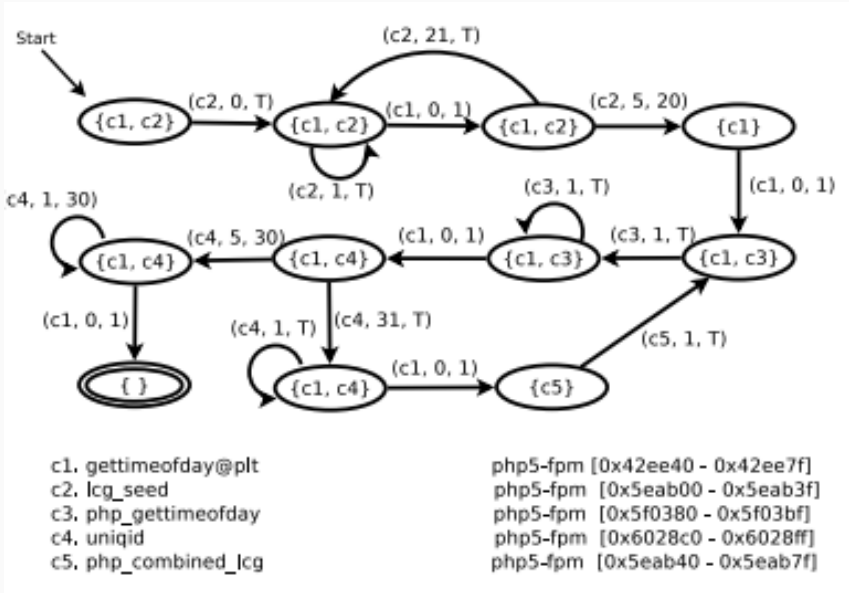

Figure 6: Attack NFA

$c2 \rightarrow c1 \rightarrow c2 \rightarrow c1 \rightarrow c3 \rightarrow$
$c1 \rightarrow c4 \rightarrow c1 \rightarrow c5$

Side-Channel Noise

- Sources of noise
  - Race conditions: two simultaneous memory loads to the same chunk
  - Unobserved multiple accesses
  - False sharing of the chunk
  - Prefetching
  - Processes other than victim using the same executable
- Solutions
  - Avoid monitoring frequently accessed cache blocks
  - Select an appropriate FLUSH-RELOAD interval
  - Avoid monitoring chunks that crosses basic block boundaries

Detection of Co-location

Co-location in PaaS

- Two step verification
  - Attacker launches many instances in the cloud
  - Each of these launched instances performs co-location detection
- For co-location detection
  - Attacker sends queries to the victim application
  - All launched instances monitors the executed code on victim side with FLUSH-RELOAD attack

Considerations

- To reduce false positives, rare events should be monitored
  - Select uncommon paths
  - Example: failed login attempts
- Attacker can avoid false positives and false negatives by attempting multiple trials
- PaaS cloud services tend to schedule applications with same runtime environments to the same machines
- Adversary already knows the runtime environment

Co-location Experiments

- IP addresses of the victim (if it is allowed by the cloud provider) can help the attacker
- Co-location was obtained in less than 3.2 hours and with zero cost

	Trials			Miss Detection	
	1st	2nd	3rd	FP	FN
DotCloud	≤ 10	≤ 10	≤ 10	0.00	0.03
OpenShift	98	120	5	0.00	0.49

Figure 7: Number of trials before co-location

Case Studies

Case Studies

- Inferring Sensitive user data
  - Cross-site request with FLUSH-RELOAD side channel
  - Inferring the number of distinct items in users shopping cart
- Password reset attacks
  - Exploits pseudorandom number generators (PRNGs)
  - Their goal is to trigger a password reset request and use the provided framework to generate time dependent url provided to user
- SAML-Based single sign on attacks
  - Bleichenbacher[4] attack to allow decryption of target ciphertext
  - Detect padding errors (previously a timing attack)
  - With NFA, attacker can monitor the code piece that performs RSA padding check

[4] Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1

Discussion

Extensions

- Extending attacks for IaaS clouds: enabled by page deduplication and shared executables between tenants
- Multiple victim copies serving the same web application: attacker needs to determine which computing instance is executing the target code
- MySQL query detection: monitoring of client MySQL library can enable inferring executed SQL queries

Countermeasures

- Mitigating side channels through program analysis
- Disabling clflush: requires hardware changes, compatibility issues
- More background noise: sharing LLC with more applications
- Restricted resource sharing: disallow sharing of memory pages
- Detecting Flush-Reload attacks

Questions?