**Summary (Please summarize the paper briefly.)**

**a)** The paper presents Shuttle, an intrustion recovery service for PaaS cloud platforms. The proposed solution is based on a record-and-replay approach, which allows to undo tained actions without loosing the effect of ligioperations.

**b)** The paper describes an approach for intrusion recovery in PaaS based applications that use distributed NoSQL-based databases as underlying storage. The approach consists of two phases. During normal execution, all user requests are logged and checkpoints are created. To obtain consistent shapshots, request splitting technique is introduced. Once an intrusion is detected, Shuttle recovers application state by removing attacker-caused state changes and replaying non-malicious requests. To parallelize the recovery process, data dependency graph is created that allows to separate requests into multiple independent clusters; such clusters can be processed in parallel. Additionally, selective replay can be carried out, where only requests affected by malevolent state changes (i.e., tainted requests) are replayed. To avoid downtime of user applications, git-like branching is used, where the replay is performed on a separate branch. Once the recovery is done, this branch is used to serve clients requests. The evaluation shows small overhead (13%-20%) both in throughput and latency.

**c)** Shuttle recovers from intrusions in PaaS applications by reverting undesired changes to early snapshots and replaying desired operations. It logs requests and responses at the HTTP and database connection layers.

**d)** The paper presents a service that helps recovering the state of a web application in case of intrusions that have destructive effects. The service records all the HTTP(S) requests, and, in case of intrusions, reverts the application to a previous (pre-intrusion) state and then replays the legitimate requests from that point on.

**Positives (What are the most important reasons to accept this paper, in order of importance? Say whether the positives dominate the negatives (1-3 sentences).)**

**a)** The paper addresses an important problem, namely intrustion recovery in PaaS platforms. The paper presents the design, implementation and experimental evaluation of a novel automatic intrusion recovery system. The paper is well organized and easy to read.

**b)** The approach presented in the paper contains some clever tricks and interesting ideas. The paper is well written, with only minor issues. Evaluation is comprehensive, showing reasonable overhead in terms of throughput and latency. The authors also provided examples of several attacks and evaluated monetary costs of using such system.

**c)**

- It develops a simple model to develop taint dependency for database requests

- It develops a record and replay framework for user requests

- It provides online intrusion recovery without downtime

**d)** The idea of a recovery process that doesn't need downtime of the app is interesting. The approach of recording the user actions and replaying them automatically if necessary is convincing. The approach is validated.

**Negatives (What are the most important reasons NOT to accept this paper, in order of importance? (e.g., the paper has serious technical mistakes, isn't original, isn't appropriate for the conference, etc.) If the overall conclusions are still likely to hold despite these flaws, please say so. Say whether the negatives dominate the positives. (1-3 sentences))**

**a)** A quantified evaluation of accuracy, with appropriate metrics and figures, is missing.

**b)** The paper falls short in providing implementation details, needed to better understand how the approach works in practice. Some assumptions (e.g., single load balancer) are questionable.

**c)** It is hard to conceive that Shuttle is protected from external channel attacks such the shellshock ssh vulnerability described. In such scenarios, intruders can destroy Shuttle recovery data. - Evidence is needed to show that this approach works with real applications. One of the challenges with real-world applications is in associating database queries with requests, especially in multithreaded environments.

Taint can propagate across requests that to not access the same database fields. Consider first a request that reads a tainted value and returns it to the user. The user may at a later time make another request using the tainted value to update an unrelated field in the database.

For some of the examples used to motivate the work, it is not apparent how Shuttle would help. Code Spaces attack on SaaS service provider infrastructure which used an external channel to delete all data from storage system (below the provider's application and database layers). The hacker accessing user data (Sec III) is an example of a privacy violation for which Shuttle is of little use.

As presented, the dependency graph rules appear to be incomplete. Alone, they are incapable of capturing simple dependency such as: Write1(A)->Read1(A)->Write2(B) where B is computed from result of A and each operation is in a separate request.

Some discussion of the impact of undoing actions with external visibility (e.g. users have viewed tainted data and performed actions elsewhere) can improve the paper, particularly in critical applications such financial applications.

Many existing cloud applications use relational databases. You should discuss how to support such DBMSes, especially in light of the fact that (as mentioned in V B) SQL dependency graphs produce many false positives

**d)** The paper presents high level elements of this recovery process, but the details of the process are missing. More important, the paper does not tie together all the components, to give the reader an un-ambigous picture of how the recovery process happens.

**Feedback to the author(s)**

**a)**

The paper interestingly presents the design, implementation and experimental evaluation of Shuttle, a novel automatic intrusion recovery system.

Shuttle is evaluated in terms of accuracy and performance.

Shuttle accuracy evaluation should be precisely quantified with appropriate metrics and figures. Shuttle is implemented as a software prototype, and is integrated with AppScale and Voldemort PaaS platforms.

How generic is Shuttle? And what is the effort required by a software engineer to integrate Shuttle to an existing PaaS system?

**b)**

The paper makes a good case providing examples of recent intrusions (Code Spaces) and attacks (Shellshock). It is easy to read, it is very well structured. I particularly enjoyed reading examples of different attack scenarios.

Some comments/questions:

- I'm concerned with the single load balancer assumption. I believe for high-load applications it very likely will not hold, and this is one direction that should be further investigated in order to make the approach more practical.

- To help the reader to understand the approach better, it would be beneficial to have a figure with an example of operation lists, version lists, different data items and versions.

- How is elasticity of the underlying database supported? I.e., what happens if a new instance is added/removed, causing the data to be moved..?

- Implementations details should be presented in more detail. In fact, implementation section is completely missing, except for stating the number of lines of code. While the general architecture is well described, I have some open questions:

- How multiple versions of data are stored? Do you assume that it is supported by the underlying storage engine or the db proxy modifies requests in some way to support versions?

- Checkpointing procedure is also not very clear. Is it performed using facilities of the db engine or Shuttle provides such capabilities? Where checkpoints are stored?

- In Section V-B, it is mentioned that for deleted data items an operation list is kept. Is it garbage collected at some point or is stored forever?

- Some questions to evaluation:

- Figure 6. Why with small throughput Shuttle has smaller latency?

- Figure 7. How many clusters were identified? What is the distribution of requests between the clusters?

- For the storage overhead, it would also be interesting to see the size of the user's data for comparison.

- - Clustering approach seems to be a nice idea, and has great performance. However, as the paper mentions, new dependencies can be created while replaying, and it seems to be a big limitation. What happens if such a case occurs?

- The mechanism to deal with delayed updates of older snapshots (last paragraph in Section IV) should be explained better. Namely, the purpose of the flag is not clear.

-Minor notes:

* There should be an arrow from Interceptor to Shuttle Storage, since the interceptor stores the log of accessed data items there. Also, since the storage is scalable, there should be a similar arrow as for, e.g., replay instance.
* Complex queries on a relational database may lead to false negatives, for instance when a read operation would have been executed on a deleted data item if this data item had not been deleted before the request execution [19]." - this sentence is not obvious, please consider reformulating.

Typos:

* P. 3 Logs every HTTP user requests -> Logs every HTTP user request
* P. 4 replays a requests -> replays requests instant SID read a version -> reads a version
* P. 7 In addiction -> In addition P. 8 Shellsock -> Shellshock

**C)** - Figure 3 needs an introduction that explains what the various components in it represent.

**d)**

- the service adds a new HTTP header to the requests, that help track the requests in the application. Could this header be exploited by some malicious user in order to break the service?

- It is unclear if the service is supposed to be autonomic or not. If it's autonomic, parts of how the service works are missing. If not, you have to clearly specify what the human operator should do, and what is the service responsibility.

- Blind writes (section V.B): how are these identified? You can have a request reading an item and in a subsequent request you write it. Do you keep track of user sessions?

- Is this a service that should be used "as it is"? Or should the user inject some code in it? The section V.F suggest that the user should inject some code, but it's not clear where (in what component from fig 1). Also it's not clear if it's mandatory or optional.

- How the whole process starts? It seems that the recovery process is triggered from the outside of the service, but I didn't find where is explicitly stated (is this one of the Manager responsibilities?).

* What happens after the initial trigger? There is no sequence of steps that ties everything together (the steps in section V.E seem incomplete: who makes and maintains the dependency graph? who determines the intrussion requests? when clustering comes into play? the clusters look like "connected components" from graph theory - is it correct? how is rejuvenation integrated here? or recovery in runtime?).
* Section V.E, step 6 - what is R.9? It's not in fig 4.
* A reader has to infer too many things that should be explicitly stated. Or has to fill the blanks:
* Section V.F: what does it mean "return an error or a response that is different"? You are refering to the HTTP response codes, or the response body of the HTTP reply (or the headers?)? A web page that contains the current date/time will be different every time is requested. Is this considered inconsistency?
* Section V.E: it's not clear when I should choose full or selective replays (altough I get a hint in VI.B-Sofware vulnerability & external channel)
* The Manager is supposed to do what's described in section V (?) Sometimes you say "tenants do this" or "Shuttle does that", but the manager does not seem to do anything. When you say "Shuttle does that" you mean "the Manager does that"? To me the manager seems like the central piece of Shuttle, but is largelly ignored in the paper.
* Should add a sections with limitations where all limitations are presented in a concise way. Now the limitations are spread all over the paper (sometimes not explicitly stated as a limitation), making it difficult to evaluate if Shuttle can be applied to a specific web application.