DENIM GROUP

the leading secure software development firm

What Permissions Does Your Database User REALLY Need?

Dan Cornell
CTO, Denim Group
@danielcornell

My Background

- Dan Cornell, founder and CTO of Denim Group
- Software developer by background (Java, .NET, etc)
- OWASP San Antonio, Global Membership Committee

Denim Group Background

- Secure software services and products company
  - *Builds secure software*
  - *Helps organizations assess and mitigate risk of in-house developed and third party software*
  - *Provides classroom training and e-Learning so clients can build software securely*
- Software-centric view of application security
  - *Application security experts are practicing developers*
  - *Development pedigree translates to rapport with development managers*
  - ***Business impact: shorter time-to-fix application vulnerabilities***
- Culture of application security innovation and contribution
  - *Develops open source tools to help clients mature their software security programs*
    - *Remediation Resource Center, ThreadFix*
  - *OWASP national leaders & regular speakers at RSA, SANS, OWASP, ISSA, CSI*
  - *World class alliance partners accelerate innovation to solve client problems*

Who has deployed a web application to production attached to its database as the “sa” or “root” user?

LIARS!

The Weakest Link

Web Application Database User Permissions

- Data = Value
- Web Applications Are Front-Ends For Web Databases
- Web Applications Are Full of SQL Injection Vulnerabilities

- Therefore: Choosing You Web Database User Permissions Has a Large Potential Impact On Your Security Posture

Problems With Web Database Access Security

- Nearly all applications use a single database user to access the database
  - *Masks the true identity of the caller to the database*
- Too often this user is hyper-privileged
- Why?
  - *Lazy configuration management for production environment*
  - *DBA attitude of "one app – one schema – one user"*
  - *"Too hard" to figure out what permissions are needed*
  - *Schema ownership required by 3rd party code*

Result

- Any SQL injection vulnerability exploit owns the entire database
  - *Schema: Map it out*
  - *Data: INSERT, UPDATE, SELECT, DELETE*
- Whole "Confidentiality, Integrity and Availability" thing: out the window
- This can even be automated:
  - *sqlmap: http://sqlmap.sourceforge.net/*
- If that database user's privileges extend beyond the database supporting the vulnerable application…

Test Environment

- (Crappy) PHP Web Application: Crap-E-Commerce
- Database Access With Full Permissions

Environment Setup Tips

- If you want to symlink to the commerce/ examples on OS X
  - http://tlrobinson.net/blog/2008/06/mac-os-x-web-sharing-apache-and-symlinks/
- Use '127.0.0.1' rather than 'localhost' for the MySQL database host
  - http://stackoverflow.com/questions/3968013/cakephp-no-such-file-or-directory-trying-to-connect-via-unix-var-mysql-mysq

What Is Wrong With Our Target Application?

- Process:
  - *Scan with OWASP ZAProxy to find vulnerabilities: http://code.google.com/p/zaproxy/*
  - *Use sqlmap to see what we can find*

- Results:
  - *Publicly-accessible SQL injections!*

Sqlmap Results

- Command
  - *./sqlmap.py -u http://localhost/~dcornell/commerce/order.php?order_id=1 --dump-all*
- Data retrieved:
  - *All of it…*

Actual Business Impact

- From sqlmap: Lost all data in the database:
  - *Usernames and passwords*
  - *Order history*
  - *Full credit card information*

- Additional possibilities: UPDATE, DELETE, INSERT

We Need To Make Some Progress

That Was With a Powerful Database User

So what happens if we deploy the application with a less powerful user?

To do this we need to know what access a legitimate user needs…

What Privileges Does a Database User Need?

- Ask the development team
  - *Good luck with that*
  - *Do they even know given frameworks and abstraction layers like ORMs*
  - *Doesn't scale*
- Ask the DBA
  - *Double good luck with that*
  - *Doesn't scale*
- Inspect the code
  - *Ugh*
  - *Error prone*
  - *Doesn't scale*

Any Way To Automate This?

- Interesting Article:
  - *http://www.teamshatter.com/topics/general/team-shatter-exclusive/what-are-my-users%E2%80%99-effective-database-privileges/*
  - *See http://www.petefinnigan.com/tools.htm for more along these lines*
- Less than ideal
  - *What assets can this user access?*

    versus
  - *What assets does the user need to access?*
- Could be helpful determining possible impact of a breach

Other Permission Calculation Tools

- .NET Permission Calculator Tool (Permcalc.exe)
  - http://msdn.microsoft.com/en-us/library/ms165077(v=vs.90).aspx
- Stowaway (Android Permissions Calculator)
  - http://www.android-permissions.org/
- Both of these tools appear to rely solely on static analysis
  - *Makes sense from a coverage standpoint*
  - *Would be really hard for databases potentially accessed by multiple applications*

Alternate Approach

- Dynamically analyze traffic to the database server
- Use that traffic as a “representative sample” of required database access
- Create user permissions based on this
- Why?
  - *Static analysis is really hard to get exactly right – this relies on observed behavior*

sqlpermcalc

- Tool that calculates the least-privilege database permissions required to execute a given set of SQL queries
  - *Written in Python*
  - *https://github.com/denimgroup/sqlpermcalc*
- Helper tools:
  - *Start and stop MySQL logging*
  - *Capture query log from a MySQL database*
- Relies on python-sqlparse for basic SQL parsing support
  - *https://code.google.com/p/python-sqlparse/*
  - *Thanks Andi Albrecht! (http://andialbrecht.de/)*

An Aside: "Pythonic"

- Definition of "pythonic"
  - *"To be Pythonic is to use the Python constructs and data structures with clean, readable idioms"*
  - *http://faassen.n--tree.net/blog/view/weblog/2005/08/06/0*
- At this point sqlpermcalc is more … "python-ish"
  - *Enjoy ☺*
  - *Any Python gurus are more than welcome to help with cleanup…*

Support Tools

- Turn on MySQL logging with mysql_start_logging.sh
  - *Not recommended for use in production because of potential performance impact*
  - *Also we're logging to MySQL tables rather than a log file – even worse*
- Retrieve MySQL log data with mysql_get_logfile.sh
  - *Pulls queries from a given user into a local .sql file*
- Turn off MySQL logging with mysql_stop_logging.sh
  - *Stops logging*

Process

- Stop webserver
- Turn on MySQL logging
- Start webserver
- Exercise application
- Retrieve logs
- Turn off MySQL logging
- Analyze logs for permission usage

Calculating Permissions

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT Permissions

- Can control on a table-wide basis
- Can control on a per-column basis for a table
- WHERE clause will require additional SELECT permissions

- Scenarios:
  - *SELECT * FROM MyTable*
  - *SELECT col1, col2, col3 FROM MyTable*
  - *SELECT * FROM MyTable WHERE col1 = 1 AND col2 = 2 OR col3 = 'three'*
  - *SELECT col1, col2 FROM MyTable where col3 = 'three'*

INSERT Permissions

- Can control on a table-wide basis
- Can control on a per-column basis for a table

- Scenarios:
  - *Full table: INSERT INTO MyTable VALUES (1, 2, 'three')*
  - *Columns in table: INSERT INTO MyTable (col1, col2, col3) VALUES (1, 2, 'three')*

UPDATE Permissions

- Can control on a table-wide basis
- Can control on a per-column basis for a table
- WHERE clause will require SELECT permissions as well

- Scenarios:
  - *UPDATE MyTable SET col1 = 1*
  - *UPDATE MyTable SET col2 = 2 WHERE col3 = 'three'*

DELETE Permissions

- Can only control on a table-wide basis
- WHERE clause will require SELECT permissions as well

- Scenarios:
  - *DELETE FROM MyTable*
  - *DELETE FROM MyTable WHERE col1 = 1*

A Note About Wildcards

- DELETE always impacts all columns in a table
  - *Hence it only has table-level permissions – not column-level*
- SELECT and INSERT sometimes impact all columns in a table
  - *SELECT * FROM MyTable*
  - *INSERT INTO MyTable VALUES (1, 2, 'three')*
- Currently we do not "know" the actual database schema
  - *Therefore we do not know all of the actual column names*
  - *So instead we track '*' to represent "all columns"*
- This should not cause problems
  - *What we see accessed in the queries should be what we need to access*

What Permissions Are Actually Needed?

- INSERT
  - ***CommerceUser***: *email,first_name,last_name,password*
  - ***CreditCard***: *CVV,expiration,number,type*
  - ***OrderItem***: *order_id,price,product_id,product_name,quantity*

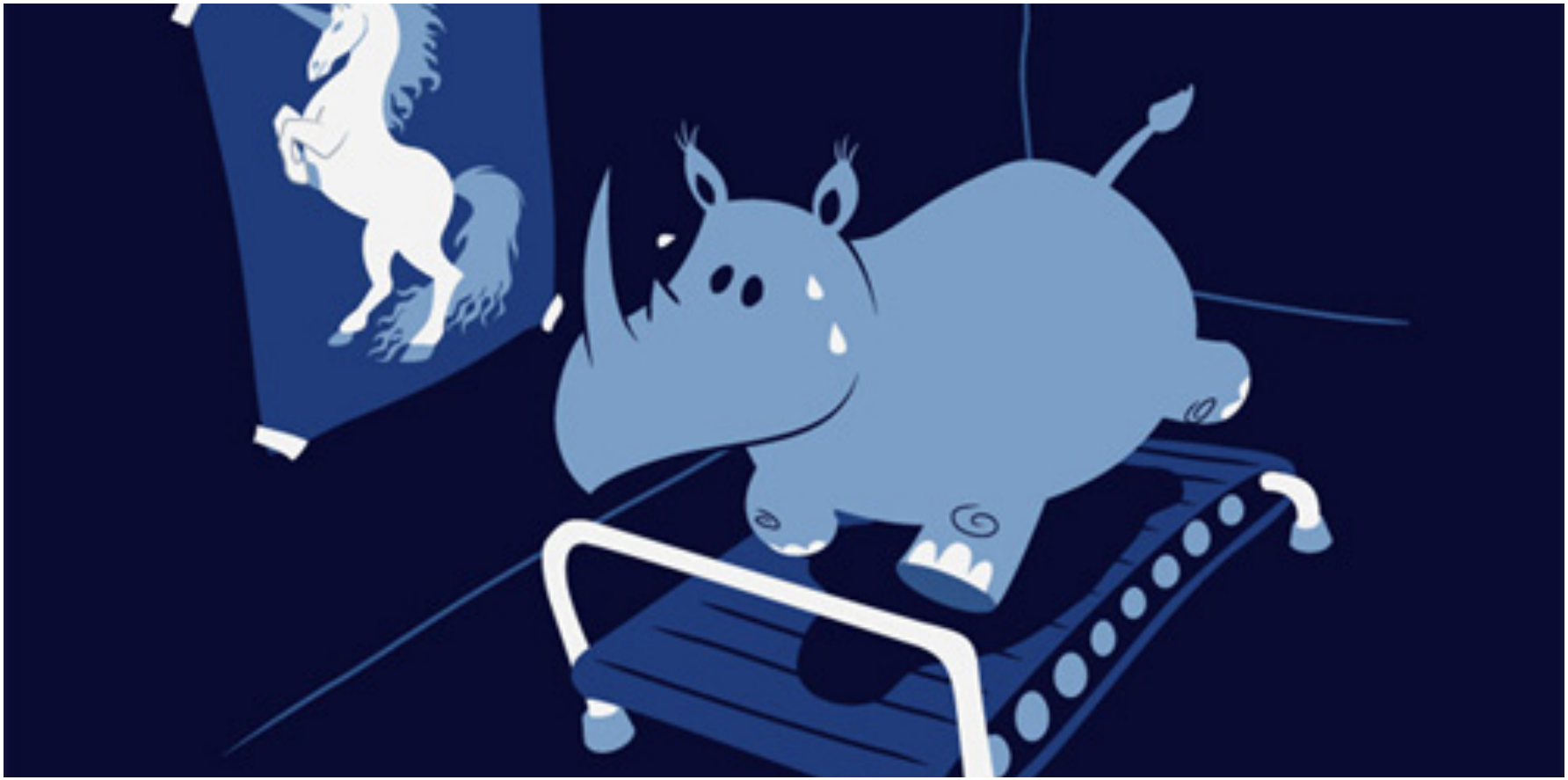
- SELECT
  - ***CommerceUser***: *
  - ***Order***: *date,total,user_id*
  - ***OrderItem***: *price,product_id,product_name,quantity*
  - ***Product***: *

Given The Model We Can Generate GRANTs

- For MySQL we need to know the user account name and host for access

```
GRANT INSERT (email,first_name,last_name,password) ON
sqlpermcalc_commerce.CommerceUser TO 'spc_publiclow'@'localhost';
GRANT INSERT (CVV,expiration,number,type) ON sqlpermcalc_commerce.CreditCard
TO 'spc_publiclow'@'localhost';
GRANT INSERT (order_id,price,product_id,product_name,quantity) ON
sqlpermcalc_commerce.OrderItem TO 'spc_publiclow'@'localhost';
GRANT SELECT ON sqlpermcalc_commerce.CommerceUser TO
'spc_publiclow'@'localhost';
GRANT SELECT (date,total,user_id) ON sqlpermcalc_commerce.Order TO
'spc_publiclow'@'localhost';
GRANT SELECT (order_id,price,product_id,product_name,quantity) ON
sqlpermcalc_commerce.OrderItem TO 'spc_publiclow'@'localhost';
GRANT SELECT ON sqlpermcalc_commerce.Product TO 'spc_publiclow'@'localhost';
```

Impact of Slimmed-Down Permissions?



Re-Run sqlmap

- Can still recover a whole lot of data
  - *But not the credit card data (or even the credit card primary key IDs)*
  - *So that is better. Kinda*
- But…
  - *No UPDATE or DELETE access to* any *tables*
  - *Limited INSERT access*
- What Does That Get Us?
  - *Can't INSERT into Products or modify Products*
  - *Automated SQL worms can't "infect" the site with malware via SQL injection*
  - *So that is definitely better*

Other Uses

- Insight into database usage
  - *Do you have any idea what database assets your web application touches?*
  - *Even if you don't generate new user permissions, you can still use this to explore*
- Forensic review over time
  - *Gather usage logs from production servers at intervals?*
  - *Why did this app suddenly start using additional permissions?*
- Compare multiple user roles or applications
  - *What does each need to do?*
  - *How are the access needs different?*

Calculating Permission for Multiple Scenarios

- Hosting Multiple Applications Accessing the Same Database(s)
  - *Two applications (public and admin) share several databases*
  - *Public site is read-only and heavily cached*
  - *Admin site is read/write*
  - *During series of attacks we had to manually calculate constrained permissions*

- Hosting Same Application In Different VMs
  - *Cannot make code changes but need to harden infrastructure*
  - *Host different configuration files for database access*
  - *Example: Falling Rock Networks Armored Stack infrastructure*
    - http://www.fallingrocknetworks.com/armored-stack.html

Limits of This Approach

- Assumes that assets touched during a test run are all that a legitimate user session will ever need
  - *If we miss something we will see runtime errors*
  - *Likely needs re-calculation when code is changed*
  - *Comprehensive unit/regression test suite can help (Rugged DevOps!)*

- Many applications require a lot of access so the security benefit might not be as great as desired
  - *In the example application: we still lost usernames/passwords*

Current sqlpermcalc Limitations

- Only supports basic SQL functionality
  - *SELECT, INSERT, UPDATE, DELETE*
- Parsing is still crudimentary
  - *More advanced SELECT statements – JOINs, subqueries – are not yet supported*
  - *Precludes use for apps using common frameworks and tools*
- Only tested on MySQL
  - *Every databases SQL dialect is a little different*
  - *Every database has different ways to grant/revoke privileges*

Next Steps

- Improve the SQL supported by the parser
  - *Support all SQL queries generated by Hibernate for a non-trivial application*
  - *Look into adding support for stored procedures*
- Clean up code
  - *This is kind of "scripty" right now*
  - *Allow others to use the capabilities*
  - *Make it more Pythonic*
  - *http://kennethreitz.com/repository-structure-and-python.html*
- Support for other databases
  - *Pull MS SQL Server queries from the Profiler*

Other Stuff To Look At

- SE PostgreSQL: https://code.google.com/p/sepgsql/

Get The Code

- sqlpermcalc on Github: https://github.com/denimgroup/sqlpermcalc
  - *sqlpermcalc Python code*
  - *Example Crap-E-Commerce app*
  - *Support scripts for MySQL*

Conclusions and Questions

Dan Cornell
dan@denimgroup.com
Twitter: @danielcornell

www.denimgroup.com
github.com/denimgroup/sqlpermcalc
(210) 572-4400