

# HAKING

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

## SECURE CODING

HOW TO FIX VULNERABILITIES  
IN REAL-LIFE APPS

SECURE CODING PHP

SECURE CODING IN DATABASE

MOBILE AND TABLET APP CODING

Vol.6 No.9  
Issue 09/2011(45) ISSN: 1733-7186

PLUS

'HOW TO' ARTICLE: SECURELY STORE YOUR PASSWORD  
REVIEW: PASSWARE KIT 11.00



# It's here! Penetration testing for Students



**Click here  
To enter the  
early bird list**

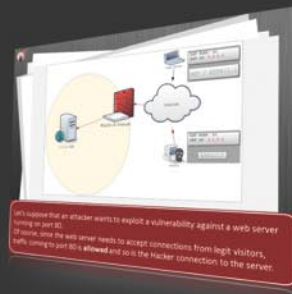


## 80% of beginners remain beginners or give up completely

We know the pain of being a beginner. You either don't have the foundational skills or you don't have a clear path to follow. Don't give up. There is a better way. Our course will teach you basics of networks and web apps.

## It's not just about 1337 instructors

Expert teachers hardly remember what took them to the expert status. It's a fact. There is no way to effectively teach beginners other than help them building strong foundations and showing them the correct path.



## You can do it

If you keep studying without a clear learning path you are probably wasting time. Secret is path and perseverance. Better a single step in the correct direction than 10 random steps. Our course will save you months of struggling and frustrations.

# You gotta see this.

[www.elearnsecurity.com](http://www.elearnsecurity.com)





Still hacking virtual machines?



**Coliseum Lab is here!**

The most epic web app hacking lab  
you have ever seen

**CLICK HERE**

14 educational challenges  
in a multi-platform  
environment.

**Epic!**

[www.coliseumlab.com](http://www.coliseumlab.com)



## HAKIN9 team

**Editor in Chief:** Ewa Dudzic  
ewa.dudzic@hakin9.org

**Managing Editor:** Patrycja Przybyłowicz  
patrycja.przybylowicz@hakin9.org

**Editorial Advisory Board:** Rebecca Wynn, Matt Jonkman,  
Donald Iverson, Michael Munt, Gary S. Milefsky, Julian Evans,  
Aby Rao

**DTP:** Ireneusz Pogroszewski  
**Art Director:** Ireneusz Pogroszewski  
ireneusz.pogroszewski@software.com.pl

**Marketing Manager:** Małgorzata Bocian  
m.bocian@hakin9.org

**Proofreaders:** Donald Iverson, Michael Munt, Elliott Bujan, Bob  
Folden, Steve Hodge, Jonathan Edwards, Steven Atcheson

**Top Betatesters:** Ivan Burke, John Webb, Nick Baronian, Felipe  
Martins, Alexandre Lacan, Rodrigo Rubira Branco

Special Thanks to the Beta testers and Proofreaders who helped  
us with this issue. Without their assistance there would not be a  
Hakin9 magazine.


**Senior Consultant/Publisher:** Paweł Marciniak


**CEO:** Ewa Dudzic  
ewa.dudzic@software.com.pl

**Production Director:** Andrzej Kuca  
andrzej.kuca@hakin9.org

**Publisher:** Software Press Sp. z o.o. SK  
02-682 Warszawa, ul. Bokserska 1  
Phone: 1 917 338 3631  
www.hakin9.org/en

Whilst every effort has been made to ensure the high quality of  
the magazine, the editors make no warranty, express or implied,  
concerning the results of content usage.  
All trade marks presented in the magazine were used only for  
informative purposes.

All rights to trade marks presented in the magazine are  
reserved by the companies which own them.  
To create graphs and diagrams we used [smartdraw.com](http://smartdraw.com) program  
by  SmartDraw

The editors use automatic system   
Mathematical formulas created by Design Science MathType™

### DISCLAIMER!

The techniques described in our articles may only  
be used in private, local networks. The editors  
hold no responsibility for misuse of the presented  
techniques or consequent data loss.

### Dear Readers,

This issue we dedicate to secure coding as it should be a  
starting point of every discussion about IT Security. A hacker  
existence is maintained by vulnerabilities in systems and  
applications, which are nothing else than human mistakes.  
There is a vital truth in the sentence that security starts with the  
code. For some this topic might not be as interesting as reading  
about attack methods and breaking systems down. It is indeed  
the most exciting thing to feel that the power of destruction lies  
in your hands. However, it's always good to learn something  
about the power of creation and defence methods. The best  
answer for attack is... to attack. But you need to first endure the  
attack. The better the code, the less damages and losses are  
caused by the presumptive attacker. So, this time I encourage  
you to learn something useful, which may not impress your  
friends, but will definitely satisfy your bosses.

I recommend to begin your reading with *Secure Coding: Hits and  
Misses* article written by Jorge Luis Alvares Medina. In this text  
you will find a concrete real-life examples of common software  
vulnerabilities, and will learn the best programming practices  
to avoid their occurrence. In *For My Eyes Only* Israel Torres  
demonstrates how to protect yourself against your data in your  
programs and scripts. If you want to know how to write a secure  
code in PHP and validate user input, you should get your  
attention to *Secure Coding PHP* by Rich Hoggan. From this  
text you will also learn some encryption techniques and other  
countermeasures. In *Secure Coding in Database* Steve Hodge  
will give you the knowledge about creating automatic audit  
trails for critical database tables and about creating processes  
to guard against and recover from bad data. An interesting  
piece for you might be also the article by Julian Evans: *An  
Brief Overview of Mobile and Tablet App Coding Security*. As  
usual Julian discusses different and broad views on security  
issues, this time concentrating on secure coding topic. Read  
this text and find out why code signing and sandboxing are two  
app security principles that should be proactively incorporated  
into the mobile coding development cycle.

This time quite a bit of learning, but I hope it will help you  
to improve your defence skills. For deductive entertainment I  
suggest you to have a little fun with Ali Hadi Bug Story. Don't  
miss also Drake and Mervyn columns!

Enjoy the reading!  
Patrycja Przybyłowicz  
& Hakin9 team



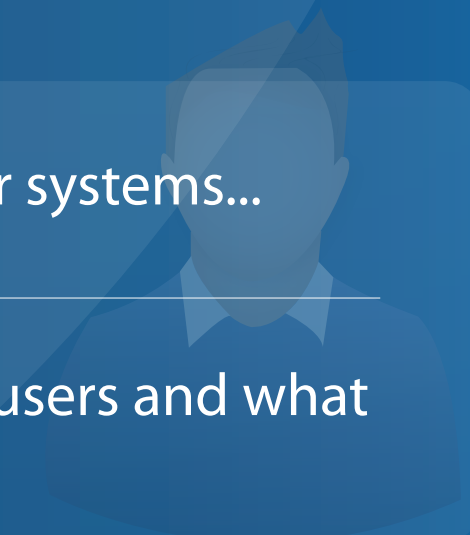
# Be reactive...

- Your systems are being attacked 24 hours a day...
- You understand the threats and are protected against them...



# Be proactive...

- My users' behaviour threatens our systems...
- I understand what motivates my users and what threats are coming my way...



ID Theft Protect provides information on threats from a user perspective.

Visit: <http://id-theftprotect.com>

## IN BRIEF

### 08 Latest News From the IT Security World

By Armando Romeo, *eLearnSecurity and ID Theft Protect*

## STORIES

### 10 The Bug Story

By Ali Hadi

Despite the fact that our Networks gardens are full of beautiful/gorgeous things, at same time they're full of bugs. The problem is that the Internet serves as connection between these gardens, which makes it easy for bugs to travel from one garden to another. A bug may be found in my neighbours' garden across the street, but in a matter of time, I will be seeing it creeping in my garden too... This time Ali tales us a quick history of the most vicious bugs in software till today. From his amusing article we will find out why software Bugs exist today and how to avoid them. He will also present us the analysis one of the most popular bug in the IT Security history and learn us on this example a quick analysis of a bug. Read the column which is as well for entertainment as for gathering some basic knowledge.

## BASICS

### 12 Secure Coding: Hits and Misses

By Jorge Luis Alvares Medina

This article expose the basics of most of the common software vulnerabilities, and explore the best programming practices to avoid their occurrence. The analysis will be made from a general perspective, but providing concrete examples and walkthroughs to clarify the concepts discussed. The examples included in each point will range from academic to real vulnerabilities found while performing different source code audits. From this article you will learn best practices that architects and developers should be aware of in order to develop applications with a proper sense of security. After the reading you will also start to think of the security analysis in terms of the actors involved, to enhance and better adapt different attack vectors the common roots of many security issues.

## DEFENCE

### 22 For My Eyes Only

By Israel Torres

Data is a marvelous thing; so easy to create but so difficult to keep track of and maintain. This marvelous thing is the very thing that can take companies down to their knees. All without anyone knowing until it is too

late... A silent killer... Data at rest and company drive shares spell disaster. Learn how to protect yourself against your data be it your programs, scripts and allow automation to occur non-interactively without you having to type your password in because you don't want to save them within the execution file. This demonstration focuses on the Apple Mac platform but can be easily geared otherwise.

### 26 Secure Coding PHP

By Rich Hoggan

It can be said that software is only as good as its code or as good as the developer who wrote that code. Yet if we used this adage to compare current web based software, we are in need of some major retrofits to the software we entrust our personal data to. The recent cyber attacks on BART – the San Francisco Bay Area's rapid transit system – only demonstrates the need for better and more secure software especially when personal and private information is at stake. As cyber attacks only seem to be growing in number, we have to start to focus more on secure coding as we try to walk the thinning line that is security and usability. With this in mind, we will discuss some of the techniques one can use to write more secure PHP code including user input verification and data encryption... In this article author shows how to write secure code in PHP and validate user input. You will also learn some encryption techniques and other counter measures.

### 32 Secure Coding in Database

By Steve Hodge

Information systems are not islands. Either data is manually entered, or, as is more commonly the case, interchanged with other systems. Some systems are very tightly integrated: a database transaction committed in one system becomes available in another almost immediately. Other systems are more loosely coupled and synchronize data on a scheduled basis. Some partners in the interchange do an outstanding job of vetting their data and making sure that the data feeds are clean. But what do you do when a data supplier comes under attack, the data becomes vandalized, or it is rendered unavailable? This text will give you the knowledge about creating automatic audit trails for critical database tables and also about creating processes to guard against and recover from bad data. You will learn building a lightweight process for rapid data recovery that avoids using complex, time-consuming database backup tools.

## ID EXPERTS SAY...

### 42 Mobile and Tablet Application Coding Security

By Julian Evans

There are practical techniques to securing app code – the first involves limiting privileges to a set of operations – this is known as sandboxing. The second technique involves identifying executables as they enter the trusted domain – aka firewall approach – do you want the app to run and how will it run are important queries. The third technique involves code trust – is the executable trustworthy? In this article author will attempt to discuss briefly some of the main mobile app security issues of today and consider what developers have to do to maintain and improve their coding security practices. Read and find out why code signing and sandboxing are two app security principles that should be proactively incorporated into the mobile coding development cycle.

## TOOL TIME

### 48 Virustotal

*By Mervyn Heng*

Hispacec Sistemas has managed the service, VirusTotal, since 1st June 2004. The VirusTotal website offers the public access to multiple Antivirus (AV) engines hosted by them to provision online scanning of individual files to uncover malware by harnessing a combination of signature-based and heuristic detection. This is the short column where you will find description of this very popular tool. If you haven't come across the VirusTotal yet this text should encourage you to pay more interest in it.

## (IL)LEGAL

### 50 What's Wrong With the Bible?

*By Drake*

Corporate IT security policies are often described by security professionals as „the Bible“. This comparison always makes my skin crawl, since it suggests a certain lack of imagination. But in reality, the comparison makes sense. Both interpretations were probably written a long time ago by people who hadn't met you, or by employees that faced precisely the same issues, technologies, and situations you face in your job today. More than that, both were probably written by different groups of people over time... Read the essay column in which the author deals with different legal curiosities and IT security cliches.

## REVIEW

### 52 Review of Passware Kit 11.0

*By Israel Torres*

Passware Password Recovery Kit Forensic 11.0 is a handy all-in-one package for recovering different types of passwords quickly and with ease. Be it from a Windows laptop, Mac VM, or USB stick this software raises the bar for password cracking. Read the program review and check is it worth it's price and buying.

Learn  
Web Application Security  
with...



## Coliseum

Virtual labs  
100% practical hands on  
training  
by eLearnSecurity

**FIND OUT**  
14 educational challenges

- ✓ Real world scenarios
- ✓ No set-up time
- ✓ Play on MS SQL Server
- ✓ Got stuck? We support!



### APPLE SAFARI 5.1 TO FEATURE SANDBOX TECHNOLOGY

Apple has announced that Safari 5.1 will feature sandbox technology which they claim will protect Mac OS X Lion users from web exploits. Sandboxing is something we at ID Theft Protect continue to promote, primarily because all processes remain isolated from core Mac and PC system files, but more importantly for us, we believe it should be installed as part of an operating system install. Safari 5.1 along with Google Chrome (as does Adobe Reader X) now both feature sandboxing technology, although unlike Chrome, Safari 5.1 sandboxing is limited to Mac OS X Lion only. For those of you who use Mac OS X, there is already a kernel-based sandbox for core processes, but sandboxing features have been significantly improved with the Safari 5.1 release.

Source: ID Theft Protect

### FACEBOOK TO INTRODUCE PASSWORD RESET

Facebook is planning to add a new security feature that allows users to reset their password so they can regain access to their accounts. The mobile password reset, will provide mobile users with the ability to identify their accounts and choose which email address should receive the password recovery URL. Facebook also indicated that they will offer additional ways for users to confirm their identity.

Source: ID Theft Protect

### FAKE GOOGLE+ APP IN CIRCULATION

A fake Google+ app has been circulating on Facebook which claims it is inviting you to join the beta Google+ project. *Liking* the app, called Google Plus Direct Access, will let scammers send you emails post content in your name and access your personal information.

Source: ID Theft Protect

### UK IDENTITY FRAUD UP 11 PER CENT IN 2011

A recent report (Fraudscape Bulletin) from the UK fraud prevention agency CIFAS, claims that identity fraud has increased by 11 per cent between January and the end of June this year. CIFAS recorded 46,609 cases of identity fraud in the last six months of 2010, compared to 51,796 in the first half of this year (2011). This increase is possibly linked to the current economic situation. People have lost their jobs, seen pay cuts, pensions frozen and more and more people are struggling to make ends meet with the ever growing cost of living – some people are also turning to mortgage fraud – desperate times

require desperate measures. There is also the small matter of over 1m 18-24 year olds who are yet to find work, which will no doubt drive some youngsters to crime and identity fraud. The banks are not lending, so this adds yet more opportunity for budding fraudsters to look to misuse other people's personal details to commit fraud.

Source: ID Theft Protect

### WINDOWS SOCIAL ENGINEERING MALWARE IN THE WILD

A recent report (Fraudscape Bulletin) from the UK fraud prevention agency CIFAS, claims that identity fraud has increased by 11 per cent between January and the end of June this year. CIFAS recorded 46,609 cases of identity fraud in the last six months of 2010, compared to 51,796 in the first half of this year (2011). This increase is possibly linked to the current economic situation. People have lost their jobs, seen pay cuts, pensions frozen and more and more people are struggling to make ends meet with the ever growing cost of living – some people are also turning to mortgage fraud – desperate times require desperate measures. There is also the small matter of over 1m 18-24 year olds who are yet to find work, which will no doubt drive some youngsters to crime and identity fraud. The banks are not lending, so this adds yet more opportunity for budding fraudsters to look to misuse other people's personal details to commit fraud.

Source: ID Theft Protect

### Microsoft release 13 security bulletins

Today (August 9th) Microsoft released 13 security bulletins, two rated Critical, nine Important and two Moderate. These bulletins address 22 unique vulnerabilities in Internet Explorer, Microsoft .NET Framework, Microsoft Developer Tools, Microsoft Office, Microsoft Windows.

Particular attention should be given to the following Microsoft security updates:

- MS11-057 (Internet Explorer). This security update resolves five privately reported vulnerabilities and two publicly disclosed vulnerabilities in Internet Explorer. The most severe of these vulnerabilities could allow remote code execution if a user views a specially crafted webpage using Internet Explorer. Microsoft is not aware of any attacks leveraging the vulnerabilities addressed in this bulletin.
- MS11-058 (DNS Server). This security update resolves two privately reported vulnerabilities in Windows DNS server. The more severe of these vulnerabilities could allow remote code execution

if an attacker sends a specially crafted Naming Authority Pointer (NAPTR) query to a DNS server. Servers that do not have the DNS role enabled are not at risk.

Specific priority should be given to the 'critical' bulletins that affect IE 6 through to IE9 on Windows 7, Vista, XP, 2003 and 2009.

Source: ID Theft Protect

## BANK OF AMERICA DATA OWNED BY WIKILEAKS? DELETED

Late last year, news about an upcoming groundbreaking release from WikiLeaks appeared all over the web. Julian Assange's organization was about to make tons of *embarrassing* documents belonging to Bank of America.

The release, that should have proved an *ecosystem of corruption* was due on January 2011 and announced by Assange in November of the same year. The same interview that begun the hunt of the WikiLeaks founder, culminated with his arrest on December 7.

So where are those documents now? According to WikiLeaks, through Twitter, Daniel Domscheit-Berg, former member and spokesman of WikiLeaks has irrevocably deleted the only copy of documents available regarding Bank of America (over 5 gigabytes), the entire US no fly list and 20 neo-nazi groups. The news appeared on the Spiegel Online, a German magazine.

According to WikiLeaks, the man had tried to extort money from the organization and for this reason was suspended in August 2010.

Source: Armando Romeo,  
[www.elearnsecurity.com](http://www.elearnsecurity.com)

## BOTNETS ON THE HUNT

Imperva's latest monthly trend report shows a massive growth in Google Hacking. Botnets are now being used to automate searches for vulnerable websites. These botnets use specific search strings to isolate websites with vulnerable code and the botnets can generate more than 80,000 queries per day. This allows attackers to draft a larger map of possible targets and puts their information gathering on a whole new level. Due to the distributed nature of the botnets, it is extremely difficult for search engine providers to block these searches as each is generated from a different IP address.

Imperva's Application and Defense Center followed a specific botnet and tracked its searches over a period of time. The botnet was discovered to conduct very wide searches for vulnerable websites and distribute the results over the botnet. The attackers

then modified the botnet to launch crafted scripts against the web sites allowing them to infect victims, compromise data or farm sensitive information on a massive scale.

Source: Armando Romeo,  
[www.elearnsecurity.com](http://www.elearnsecurity.com)

## LATEST NEWS FROM THE WAR TO SPAM TRENCHES

There have been many surges in spam as it hit a two year high in the past couple weeks. Researchers at M86 Security have tracked the spam and suggested it is attackers seeking to rebuilt their botnet armies. The main botnet at work is the Cutwail botnet which accounted for 13% of all spam at the beginning of their monitoring period and ending accounting for 24%.

Over two years ago, the Rustock botnet was taken down and spam fell to 72.9% of total internet traffic. Through botnet takedowns and arrests, spam traffic dropped over the next two years. Recent surges suggest that attackers are preparing to fight back and rebuild their armies for malicious uses.

Most of the spam delivers the malware via attachments to the email. The attachments claim to be a note or in voice within a ZIP file but also contain additional fake antivirus programs and spyware.

Source: Armando Romeo,  
[www.elearnsecurity.com](http://www.elearnsecurity.com)

## SECURE OPEN WIRELESS BY IBM

IBM is pushing for a new secure open wireless method. This new method can protect all network users from sniffing, sidejacking and man-in-the-middle attacks. It behaves much like HTTPS and uses certificates to insure the integrity and confidentiality of the data. The certificate is tied to the SSID of the network and verifies the user is connected to a trusted wireless access point.

The client doesn't need any authentication or certificate prior to connecting; the process simply verifies the SSID of the network and then creates an encrypted session between the client and access point. The network provider only has to purchase a digital certificate and incorporate a RADIUS server such as Free RADIUS. Cross of IBM states *Once people learn that they can create a secure open wireless network, I think that it's going to become an expectation. When users go to connect to a wireless network, they're going to want that wireless network to be secure.*

Source: Armando Romeo,  
[www.elearnsecurity.com](http://www.elearnsecurity.com)

# The Bug Story

Despite the fact that our Networks gardens are full of beautiful/gorgeous things, at same time they're full of bugs. The problem is that the Internet serves as connection between these gardens, which makes it easy for bugs to travel from one garden to another. A bug may be found in my neighbours' garden across the street, but in a matter of time, I will be seeing it creeping in my garden too.

---

## What you will learn...

- Quick history of the most vicious bugs in software till today,
- Why software Bugs exist today,
- Quick analysis of a bug,
- How to avoid computer bugs

## What you should know...

- Basic computer software installation or implementation,
  - Acronyms such as CIFS, NetBIOS, DCE, RPC, NDR format, Stack.
- 

They are spreading so fast that sometimes it's very hard for us to confront them. These bugs are causing security flaws to our gardens and, for sure, to the whole Internet.

Companies with big names and government agencies are being hacked because of different security bugs that exist in one of their systems. Also, if you follow any news groups such as Bugtraq, or Full Disclosure, you will be shocked with the amount of bugs that are reported on a daily basis. Bugs are everywhere, there is literally no corner in our garden without some of them. While some of them walk like a tortoise and can be crushed by our hands easily, some move so fast that it really makes you lose your brains trying to catch or confront them.

Throughout the history, our Internet gardens suffered lots of problems because of bugs. I still remember a couple of worms and viruses that did huge damage such as Melissa back in 1999, the ILOVEYOU email in 2000, Nimda in 2001, Slammer in 2003, and I don't think the world will forget Confiker 2008 for sure! These are the most vicious worms and viruses that have attacked our gardens because of a bug in our applications, systems, or even protocols.

## Are They From Outer Space?

Looking back at the attacks we mentioned, we will realise that they were not from outer space. We don't need to believe in flying sorcerers because they simply were not from Mars, Jupiter, or some other hidden

planet in the galaxy! They all came from the same place HUMAN. Yes, that's the truth about them, and believe it or not, it's us who make these bugs, and we are the ones paying their bills! The human mistakes can be classified into two categories:

*Development:* Requirements, Specifications, Analysis, Features, Design, Programming, Testing, etc.

*Execution or Operation:* Software Installation, Software Configuration, Software Usage, etc.

Imagine a public service that requires authentication and has a hardcoded username and password, or with no password at all? When I say a public service I talk about a bench in a public garden that can be used by anybody. That's an example of a bug in the development of that service. Imagine an ftp server configured to allow anonymous logins, which even allows anonymous uploads of files. That's an example of an executional or operational bug. Both types of bugs are alive due to mistakes made by us, the humans, not the Terminators!

## Once Upon A Time, There Was A Bug Called...

Confiker is really an interesting Bug Story to tell, not only because of its impact and its infection spread, but because it gives us some good clues and ideas how human programming mistakes lead to this worm crawling in our gardens. Plus, there is another interesting thing in Confiker that I will leave to be discussed at the end of story (maybe you will be already asleep and won't know anything about its ending!).



Once upon a time, there was this feature called `NetPathCompare`, which is a DCE RPC call that can be run over the CIFS protocol for the Windows Operating System's NetBIOS. CIFS is used in different network sharing (files, printers, serial ports, etc.) plus, it can be used to execute commands remotely! To use this feature and run a command via this DCE RPC, the arguments have to be encoded in NDR format. `NetPathCompare` allows a client to compare two paths, some serialization/marshalling thing. All that is needed is to log into the server and call the API to compare the two paths and the server will return the results if the path matches. Nice feature, right? Just can't imagine why people would want to use it! Now, to compare the paths, the server would use `NetPathCanonicalize` to get rid of `../../../../` things. `NetPathCanonicalize` is exported by the server too, so clients can even have their paths canonicalized by a server. It turned out that `NetPathCanonicalize` has a bug if you specify a path which gets over / like `foo/bar/../../../../` you write over the stack! And this bug is what is known as MS08-067, with the name conficker.

## A Story With No End!

So for conficker, you had everything. Sloppy specifications, sloppy implementation, a feature which was not really used or tested, and this is why this bug is still a problem after 3 years. Sorry, what did you say? You're saying that this bug story is not over? Sorry, no happy ending? Unfortunately no! There is still around 6 million nodes infected (you can always check the Conficker Working Group and their statistics for Infection Tracking, check the *On the 'Net* section for the URL). Wow. Even though Microsoft has released patches to solve this bug, the problem against these patching solution relies that conficker disables automatic patching, manual patching and antivirus scanners too. Not just that, conficker is a worm that has evolved many times. That's why researchers gave it names: Conficker A, Conficker B, Conficker C, Conficker D, and Conficker E. This bug is using different advance infection techniques which make eradicating it very hard and complicated. No, this time a big WOOOW!

## I'd Love To Have One Running In My Backyard. How?

If you want to check out this whole bug you can install the Dionaea honeypot system implemented by Markus Kotter. Markus implemented the CIFS NDR and the DCE RPC to get Dionaea capture this bug (worm). So Dionaea comes out of the box, capable of capturing conficker. All you need to do is wait for a hit on your Dionaea honeypot. I will dedicate an article in the future on installation and usage of the Dionaea honeypot, so just keep tuned with Hakin9.

## Any Electronic Pif Paf To Use?

Unfortunately there is no pif paf for these bugs. Yes, I know they live in our gardens, but as I told you, there is no 100%

### On the 'Net

- Symantec Top Computer Viruses and Worms in Internet History,
- Dionaea Honeypot System by Markus Koetter,
- Conficker Infection Tracking,
- Know Your Enemy: Containing Conficker To Tame A Malware,
- An Analysis of Conficker's Logic and Rendezvous Points.

cure for them. These bugs are found because of humans, and humans will in one way or another make a mistake. But to develop a secure code and reduce the amount of bugs in it, I think the medical prescription below will help:

- Choose a programming language that will assist you, not make your life harder. That's why I basically like python over any other programming language.
- Reducing the functionality to the minimum required is a very important point to keep in mind. The more functionality you provide, the more doors you open for a bug. And this is basically what Michel Angelo defined as a perfect back in the days when he had a stone and removed everything not required, and after he was done it was a sculpture!
- Make your software specifications clear enough for implementation. The resulting implementation can only be as good as the specification.
- Never hardcode usernames and passwords specially for a service that will be accessed through the network.
- Always read the software's configuration manual before installing it. Lots of bugs were found and exploited because of a wrong software configuration.
- Finally, never rely on your antivirus status.

### Summary

Bugs in our gardens will always be a problem, because of our human mistakes. You must be convinced that there is no system or application that is 100% bug free, and that you have to really do good testing before releasing your software. So unfortunately we will continue to see attacks and infections because of these bugs. This leads us to the conclusion that *the most secure code is code which never gets written*.

### ALI HADI

*The author has been working as a network security officer for different large companies for more than five years. His day-to-day activity is related to firewall auditing, IDS/IPS, and policy enforcement. He holds a Ph.D. degree and MS.c. degree in Computer Information Systems (CIS), and a BS.c. degree in Computer Science. Throughout his working career he managed to gain a couple of well-known technical certificates such as: OSCP, ECSA, CEH, CNI, CLP10, CLA10, CLDA, IBM Certified Specialist – System Administration, Novell Linux Specialist, and RHCE.*

# Secure Coding

## Hits and Misses

This article will expose the basics of most of the common software vulnerabilities, and explore the best programming practices to avoid their occurrence.

### What you will learn...

- best practices that architects and developers should be aware of in order to develop applications with a proper sense of security
- think of the security analysis in terms of the actors involved, to enhance and better adapt different attack vectors
- the common roots of many security issues

### What you should know...

- a general knowledge in application design and development
- basics on web based languages (Javascript) and protocols (HTTP), and SQL essentials.

The analysis will be made from a general perspective, but providing concrete examples and walkthroughs to clarify the concepts discussed. The examples included in each point will range from academic (textbook cases) to real vulnerabilities found while performing different source code audits.

Following is a list of the topics the article will cover:

- Basic principles of secure programming (including real-life examples and references)
  - Implement Security-In-Depth

- Fail in safe mode
- Follow the less-privilege principle
- Compartmentalize
- Promote privacy
- Basic failures in secure programming
  - Buffer Overflows (the basics, as this topic was already covered by Hakin9)
  - Cross Site Scripting (includes a real-life walkthrough)
  - SQL injections (includes a real-life walk-through)

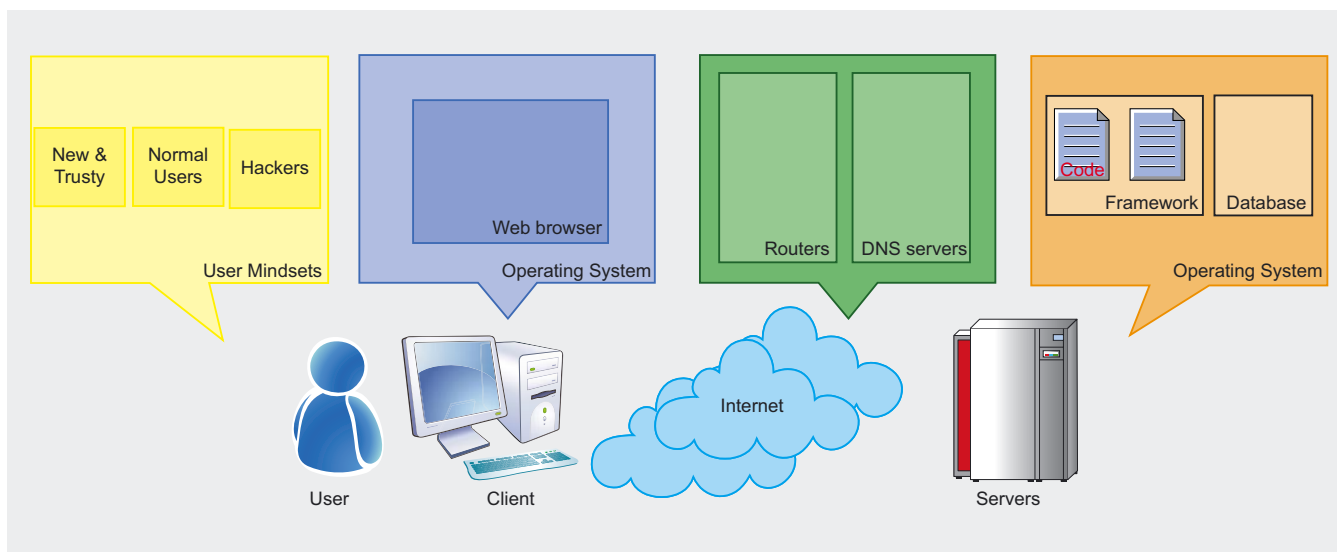


Figure 1. An application running dissected in terms of the components involved

## Basic Principles Of Secure Programming

Application security is a complex world. Every single link in the chain – the code, the application environment, the communication channel, the user environment and the user itself – poses risks that developers should be aware of while, by the contrary, reality shows that most of these security concepts are ignored.

It is a common mistake to leave the security analysis for auditors. Not only is it inefficient – sometimes it is necessary to rewrite entire blocks of code due to security issues – but also ineffective – auditors might miss bugs in their analysis, especially when there is plenty.

And contrary to popular belief, it is not necessary to be a *hacker* to write secure code. You just need to be aware of the threats that you can find in the wild, and develop applications with that broader *sense of security* in mind.

We will make our analysis in terms of the different actors involved and their interaction. For example, a web-based banking application involves several different actors: the code itself, the framework executing the code, the underlying operating system of the webserver holding all of it, the Internet, the operating system of the client computer, the web browser application, and the user itself – and that is just for a start.

Every component involved can be attacked, and result in a user account being compromised. And despite developers only having control over the code they write – and they should not clean other peoples' mess – the problem goes beyond responsibility or scope: if a customer of *Bank of America* lost his money, he will probably put their money somewhere else and encourage others to do the same, no matter whose

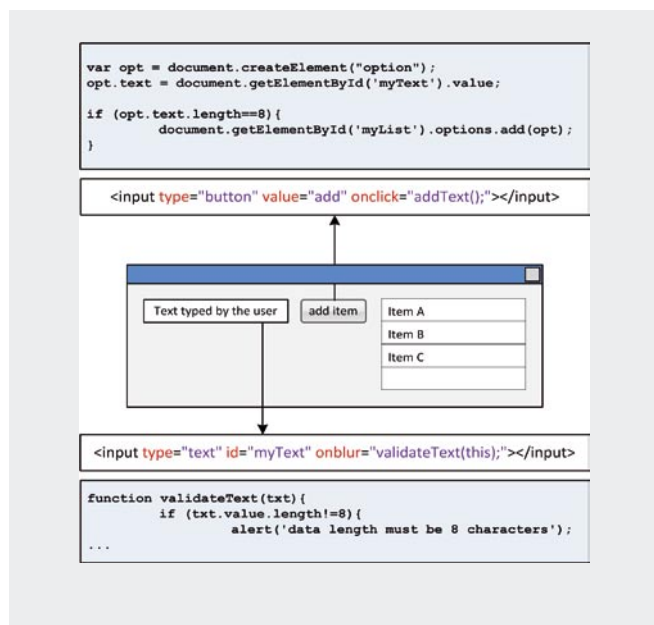


Figure 2. A page with HTML controls implementing security in depth

fault it was. Thinking in those terms, it makes sense to design applications considering potential incidents sourced elsewhere.

In this article we will analyze the security in terms of the actors described above, discussing the risks each of them is subject to and how failures in the defence of a given component may affect the others, while discussing how these risks can be mitigated from the code itself. This approach will result into merging security principles that are usually treated independently when discussed in textbooks.

## Implement Security In Depth

The basic idea is that, if a security barrier fails in stopping an attack, the next layer should be able to do it. Imagine that a user enters a value into a *textbox*, then clicks on a *button* and the value is appended to a HTML *select* containing more values that, later, will be stored in a database.

The *textbox* should validate the data entered when the `onBlur()` event occurs. The button `onClick()` event will append the data to the select list. At that point you should implement code for controlling the data that, allegedly, was verified before. This redundant protection is what we meant by security in depth.

## Fail In Safe Mode

Any system beyond a given level of complexity should be expected to fail. And, when error conditions occur and they are not controlled, information could get damaged, altered or leaked – stack traces, directory and file names disclosures, or opened references to resources – depending on the exception's nature. For example, if a routine needs to gain administrative privileges to execute a function, here is a good sample on how not to code it: see Listing 1.

As can be seen, the `unreliableCode()` function could throw an exception and the `isAdmin` flag would remain true despite the user not being a real Administrator. A

### Listing 1. An insecure privilege elevation

```

isAdmin = true;
...
try {
    unreliableCode();
    isAdmin = isUserInRole("Administrator");
}
catch (Exception ex) {
    log.write(ex.toString());
}

```



**Listing 2.** A secure way for implementing the same privilege elevation

```
isAdmin = false;
...
try {
    isAdmin = true;
    unreliableCode();
    isAdmin = isUserInRole("Administrator");
}
catch (Exception ex) {
    isAdmin = isUserInRole("Administrator");
    log.write(ex.toString());
}
```

more proper way to code it would be as follows: see Listing 2.

The code below ensures the `isAdmin` flag is set true when strictly necessary, and will have the corresponding value otherwise.

You must have these potential failures in mind when writing your code, and include routines for leaving the system in the safest state possible. Pay attention to error handlers as you do your code, and step your debugger through it several times ensuring that you hit every error handler. Also, make sure your test suites force your functions to fail, exercising every line of code. And, if detailed error messages are required for debugging purposes, ensure they are disabled in the production environment.

### Follow The Least Privilege Principle

It is all about allowing what is needed when it is needed, and nothing more. If an object only needs read permissions, do not grant write access to it. If a certain task requires high privileges to be executed, remove these privileges after the task is complete. This advice can be extended to any component in the system, experience shows that it is usually ignored by both administrators and developers at many levels: it is quite common to find file upload servlets allowed to read and write all over the file system that do not check if the file name passed contains path traversals.

Always define the security context required for your code, and grant access only to the resources it needs to work. If any part of the code requires higher privileges than usual, consider factoring that code out and running just that code with the higher privileges. For example, when a function requires higher privileges than the rest of the functions on the same application page – usually implemented as impersonation of different operating system users

– the best approach is to split the code into separate assemblies and run only the privileged code inside that privileged security context. And if any of these assemblies works with files or other resources, close their references as soon as possible because they could leak from this security context to the other. This will involve interprocess communication such as *COM* or *Microsoft .NET remoting*, and you will need to design the interface to that code to keep round-trips to a minimum, but that is a small price to pay for keeping your assets safe.

### Compartmentalize

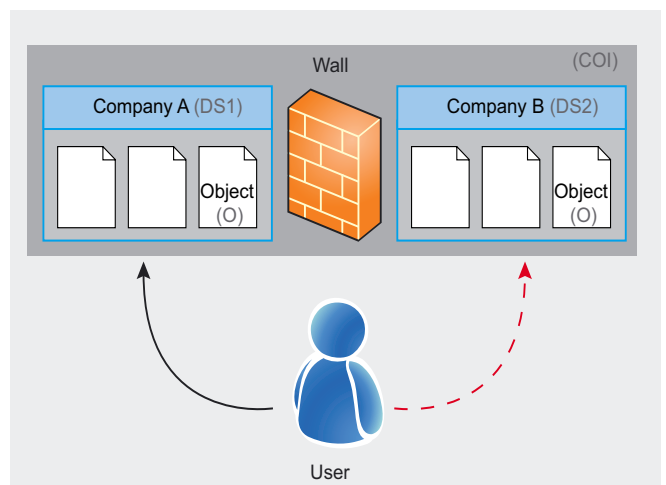
It refers to allowing access to a given element only to people that needs to access it, and nobody else. The *Unix* file DAC model is a good example on how not to compartmentalize, as it lacks key features required in modern and collaborative systems, where it is not enough allowing just a set of permissions to the owner, the group the owner belongs to, and the rest of the world. Something more versatile, more granular, is needed when working with complex business rules and permissions.

The *Brewer and Nash* model (also known as the *Chinese Wall* model) is a robust security model used for both privacy and integrity for data. It defines *subjects* as any entity accessing (by the means of read or write rules) protected objects that can be in the following hierarchy: *object*, *dataset* (DS), and *conflict of interest* (COI) classes.

A given user can access an object  $o$  as long as he has never accessed an object  $o'$  such that:

- $COI(o) = COI(o')$
- $DS(o) \neq DS(o')$

A subject associated with a given user may read the object  $o$  only if the user may read it, and may write the same object only if:



**Figure 3.** The Brewer and Nash model

- The subject may read  $o$
- The subject has never read an object  $o'$  such that  $DS(o) \neq DS(o')$ <sup>(1)</sup>

The first two conditions guarantee that a single user never breaches the wall by reading information from two different *datasets* within the same COI. The third condition guarantees that two or more users never cooperatively breach the wall by performing a series of read and write operations. Suppose that a *subject* S1 has previously read from DS1, and S2 has previously read from DS2. Consider the following sequence of operations:

- S1 reads information from an object in DS1
- S1 writes that information to an object  $O'$  in a DS from a different COI
- S2 reads that information from the same object

At the end of this sequence, S2 would have read information pertaining to both DS1 and DS2, which would violate the policies since both DS are in the same COI. But, according to the rules(1), the write operation is blocked: once a subject reads two objects from different DS, that subject may never write any object. So for read and write access, a user must create distinct subjects for each DS. For read-only access, a user can create a single *subject* to read from several COI.

Another approach is the *Role Based Access Control* (RBAC) model, designed for scenarios with a high number of *subjects* and *objects* where the number of authorizations becomes extremely large and, if the user population is dynamic, the number of *grant* and *revoke* operations to be performed becomes very difficult to handle.

In the real world, security policies are dynamic: access rights need to change as the responsibilities of users change. When a user is authorized for a system,

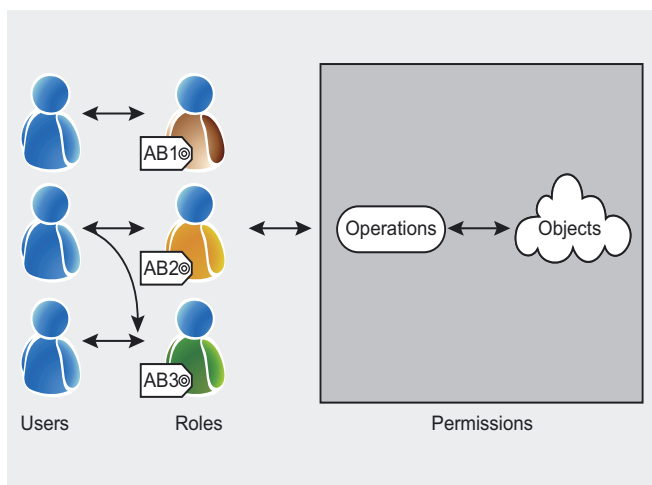


Figure 4. The RBAC model

a set of rights is established. When this user changes job functions, some rights should be deleted, some maintained, and some added.

RBAC addresses this problem by changing the underlying subject-object model. A role is a job function or title – a set of actions and responsibilities associated with a particular activity – Now, instead of an *access control policy* being a relation on *subjects*, *objects*, and *rights*, a policy is a relation on *roles*, *objects*, and *rights*; this is called *right assignment*. Further, *subjects* are now assigned to *roles*; this is called *role assignment*. Each *subject* may be assigned many *roles*, and each *role* may be assigned many *subjects*. Finally, *roles* are hierarchical. For example, the role AB1 should have all the rights the role AB2 does, and more.

Roles can be compared to groups in *Unix* file system DAC, with two major improvements. First, a group is a set of users, whereas a role is a set of rights. Second, a user is always a member of a group, whereas a subject may activate or deactivate the rights associated with any of the subject's roles. This enables fine-grained implementation of the principle of Least Privilege. Subjects may login with most of their roles deactivated, and activate a role only when the rights associated with the role are necessary.

Nearly all real-world systems (including most operating systems and database systems) implement some form of RBAC. Either discretionary or mandatory policies can be built using RBAC as the underlying model.

**Listing 3.** A sample HTTP response disclosing excessive information in the Server header

```
% telnet 123.123.123.123 80
Trying 123.123.123.123...
Connected to 123.123.123.123.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 30 Jun 2008 15:59:40 GMT
Server: Apache/2.2.3 (Debian) mod_python/3.2.10
        Python/2.4.4
mod_perl/2.0.2 Perl/v5.8.8
Last-Modified: Mon, 07 Jan 2008 08:00:35 GMT
ETag: "445a5-26-41450ec0"
Accept-Ranges: bytes
Content-Length: 38
Connection: close
Content-Type: text/html; charset=UTF-8
```

## Promote Privacy

The idea of encouraging privacy between users and your application goes both ways. On one hand, applications should not ask for sensitive information about their users unless strictly necessary. Reasons are quite obvious: the more of it you store, the more attackers will be tempted to target you. And whenever delicate data needs to be stored, it is recommended to keep it in a separate computer and also to store it ciphpered.

On the other hand, applications should not provide more information that necessary to their users. While not a solution, the concept of *security through obscurity* at least might help. A simple HTTP connection to a server could be quite revealing: see Listing 3.

Considering that plenty of vulnerabilities become public every day, along with the techniques to exploit them, you should keep in mind that nobody – but occasional attackers – needs to know that you are using *Apache 2.2.3* in your *Debian* server, running *Python 2.2.4* and *Perl 5.8.8*.

## Basic Failures In Secure Programming

In the application security field, there is a golden rule that supersedes every other: *the world outside your code should be treated as hostile and bent upon your destruction*. From there, it is possible to analyze it from two different standpoints: vulnerability-based and language-based security best practices.

Here we will take the first approach, which I consider more appropriate – as this is extendable to any other programming language with the same characteristics – and then season these vulnerability descriptions by adding specific examples from the real world – for obvious reasons, real company names will not be disclosed.

While not comprehensive (it would take books to cover all of the well-known security threats) this article

will discuss some of the most common security issues found in the wild, as long as the workarounds to mitigate them.

## Buffer Overflows

This magazine has invested lots of pages into explaining buffer overflows. However, an article in *secure coding* would be incomplete without covering this type of attacks. Just to be fair, we will make a quick review.

Buffer overflows have been the main software security issue for decades, as it becomes easy to make these type of mistakes with programming languages that do not have implicit protections against them. The basic idea goes as follows: in your routine, you allocate memory for different purposes, and then write data in there without checking if the memory allocated is big enough. If it is not, the payload passed will overwrite other information, such as the return address for the function. Then, different things can happen: programs may act strange, keep running, or crash.

If the execution flow can be controlled you can also include arbitrary code chunks in that oversized payload, overwrite the return address so it points within that code chunk, and voila! You have an exploit.

There are two major types of buffer overflows: stack-based and heap-based overflows. Exploitation of the second type is technically more complicated, as you need to know which variables are critical, find a memory block that – if properly altered – changes those variables, and even so those changes will probably crash the application. Nevertheless, in some scenarios – like the heap spraying technique so famous nowadays – successful exploitation can be achieved with a higher success rate.

The first line of defence is to validate every single data used as input, allowing nothing but the length and type of data needed, and considering potential ways of

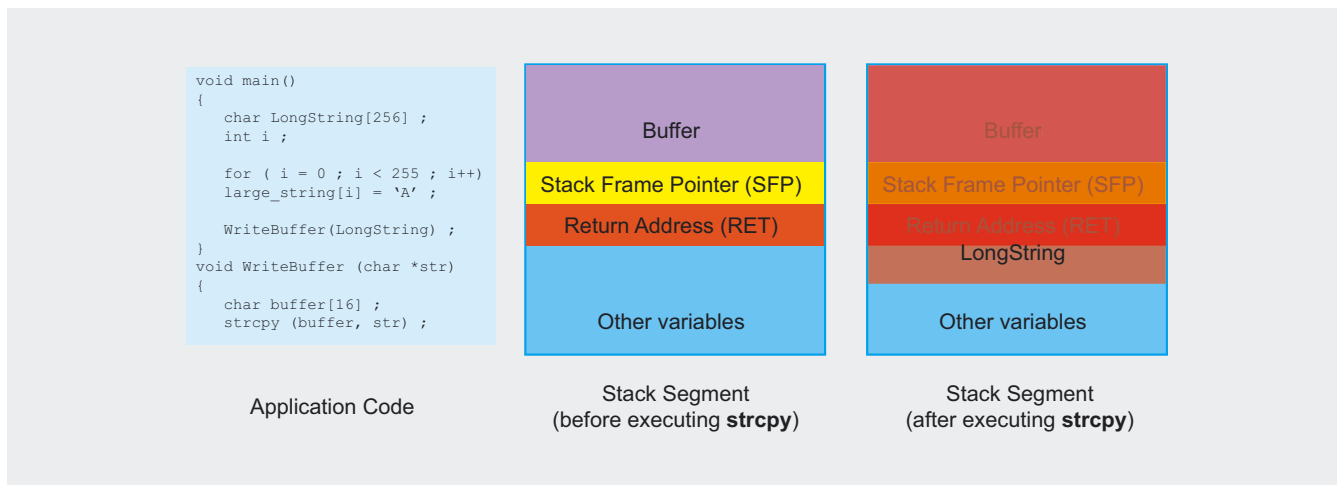


Figure 5. A function causing a stack overflow before & after it occurs

Easily add security  
at the source

**buguroo**  
offensive security

In buguroo our expert teams in security, hacking and programming allows us to find solutions to simplify the development of secure code to our customers.



“Simplicity is the ultimate sophistication”

Leonardo da Vinci

buguroo has designed and developed bugScout, a powerful managed service for **source code vulnerability analysis:**

- **Effectiveness.** bugScout automatically detects over 94% of the vulnerabilities that can be found within the source code.

- **Simplicity.** bugScout includes a project, application and analysis classification system, incorporates a reports manager and makes vulnerability management a lot easier.

- **Scalability.** bugScout works in a decentralized, cloud computing environment.

- **Parallelized.** bugScout is designed to simultaneously audit multiple source codes without affecting performance.

- **Customizable.** bugScout is a multitasking and multiuser platform providing for rights granularity. The user interfaces are completely customizable.

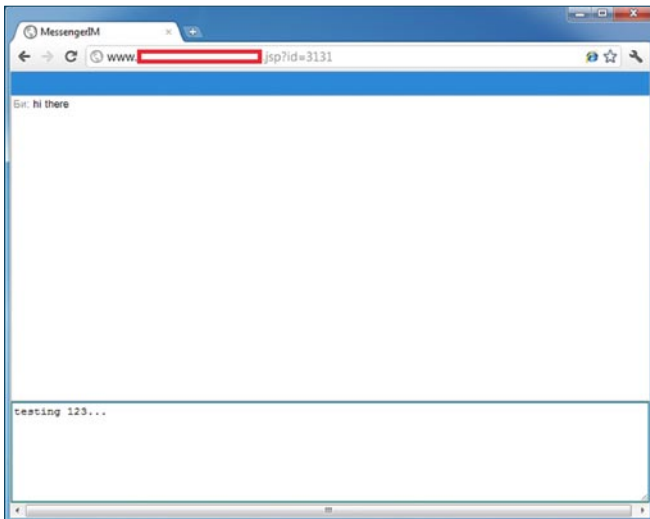


**Table 1:** C++ functions considered insecure

String functions	strcpy()
	strcat()
	sprintf()
	vsprintf()
	gets()
Scanning functions	scanf()
	sscanf()
	fscanf()
	vfscanf()
	vscanf()
	vsscanf()
	MS functions
	_tcscpy()
	_mbscpy()
	wcscat()
	_tcsctat()
	_mbscat()
	CopyMemory()
Other functions	realpath()
	getopt()
	getpass()
	streadd()
	strecpy()
	strtrns()

subverting the intended behavior of the function. Just on top of that, there are certain functions for string handling – the biggest source of buffer overflows – that should be avoided (see Table 1). And beyond the code itself, there are many other resources: non-executable stack, boundary checks at compiler level, StackGuard, ASLR, and etcetera.

The chances for a buffer overflow to occur are strongly bound to the programming language. For example, if you are using Java, you can stop worrying about this (and start worrying about other things). But if you are using as C++, there are many things that you should have in mind.



**Figure 6.** A real web-based chat application

**Listing 4.** Code snippet from the affected .jsp file

```
public void messageReceived(String msg) {

    String s = " <font color='gray'> "
        + friendUser.getUsername() + " :</font>" +
        msg + "<br>";

    out.setHTML(out.getHTML() + scanSmile(s));
    scroller.scrollToBottom();
    //startBlink();

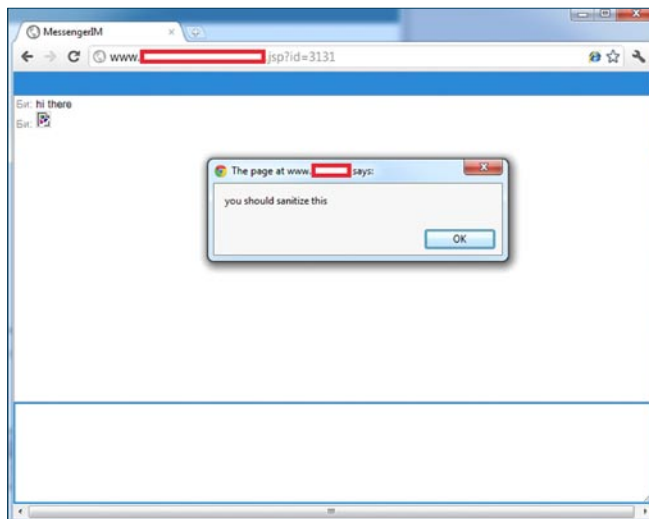
    sound.play();

    focus();
}
```

## Cross-Site Scripting

Introduced as a result of not having adopted encoding practices at the time of displaying data retrieved from insecure data containers – user-supplied, XML files, and etcetera – the risk associated with *cross-site scripting* (XSS) vulnerabilities varies according to how realistic it is for an attacker to gain control of the data being used in output.

There are two types of XSS: persistent and non-persistent: exploitation of the persistent type involves a two-step process, where the first step requires the attacker to store a malicious payload in the application's database and the second consist of having the victim access a page that displays the malicious payload un-encoded. The non-persistent version involves a single-step process where all the attacker has to do is to lure the victim into accessing a specially crafted URL.



**Figure 7.** The same application rendering the HTML/Javascript code in the messages received

To better explain the concept, we will analyze a real-life example of a web-based chat application – concealing the company identity – as can be seen in the Figure 6.

The following code belongs to the .jsp page in the URL, which handles the messages sent and received: see Listing 4.

The function `messageReceived()` takes the messages the `msg` variable, and then updates the HTML in the conversation window with the contents of that variable performing no sanitization at all. Thus, if any of the participants includes HTML code, it will be appended to the conversation and executed.

The idea behind most cross-site scripting attacks is to insert scripting code, not just plain HTML tags and therefore be able to execute JavaScript in the security domain of the vulnerable webpage. One of the advantages of belonging to the security domain of the website is having access to the website cookies, which would allow the attacker impersonating the victim.

### A Word On Fixing XSS Vulnerabilities

Encoding must be performed on every single piece of the code that displays the content of a dynamic variable whose contents were retrieved from an insecure container (user-supplied data, database, and etcetera). It will only change the internal

representation of the data so that it does not harm the context where it is being used and they will also let the data serve its original purpose at the time of displaying the contents to the user. Even though it may sound like a simple task, encoding has a catch. It is not recommended to blindly use a generic encoding method at the time of encoding variable contents. The context where the data is used is very important and given the fact that encoding will change the data, if used incorrectly not only could it open the door to other ways of exploiting the bug but also break functionality.

We highly recommend not using any kind of generic filter to prevent Cross-Site Scripting vulnerabilities. On the one hand, filters are easier to implement as they provide a generic way of treating multiple problems but Cross-Site Scripting vulnerabilities can take place in different contexts and therefore require different types of measures. On the other hand, filters can easily be bypassed unless strict filtering rules are imposed which, in most cases, end up affecting the original intended behavior of the application.

Cross-Site Scripting attacks have evolved in the past few years and so have the techniques used to disguise them. That is why creating a regular expression aimed at preventing all possible disguised attacks will definitely end up filtering non-malicious data and creating false-positives.

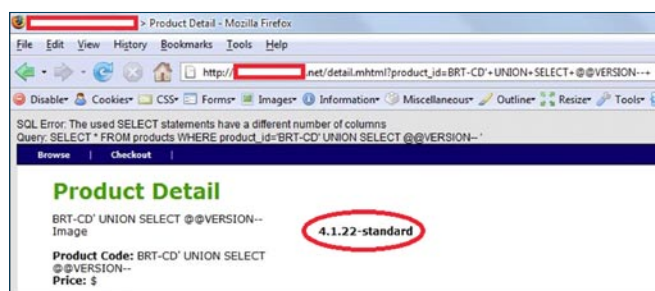


Figure 8. A SQL injection used to obtain the MS SQL version

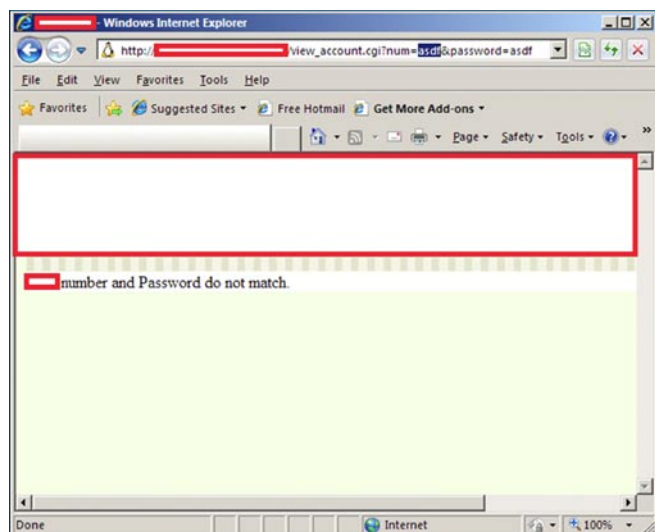


Figure 9. A failed login attempt using wrong credentials

### SQL Injections

SQL injections occur whenever input is used in the construction of a SQL query without being adequately constrained or sanitized. The use of dynamic SQL (the construction of SQL queries by string concatenation) opens the door to these vulnerabilities. SQL injections allow an attacker to access the database servers, by allowing for the execution of SQL code under the privileges of the user used to connect to the database.

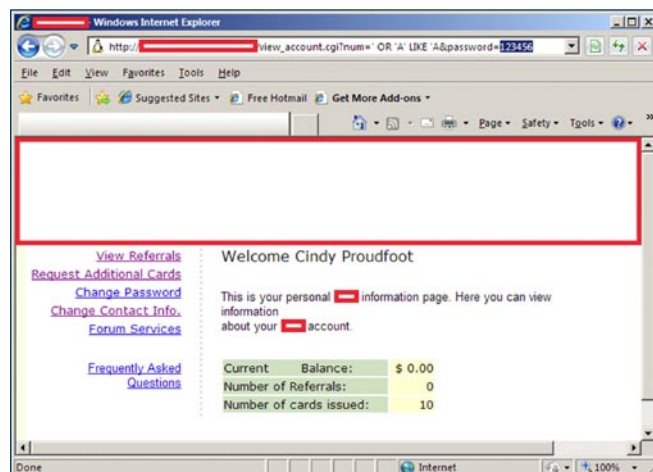


Figure 10. A successful login abusing a SQL injection

There are two types of SQL injection vulnerabilities: error-based and blind. In error-based SQL injections the error message reported by the database – under an invalid query – is displayed to the user, allowing him to leverage information based on this output.

In the case of blind SQL injections no error information is returned to the user, increasing the difficulty of detection – and exploitation – of the vulnerability. Instead, the application response differs based on a SQL condition provided by the user.

In both cases, the underlying cause is the same: the application builds the query on the fly using parameters provided by the user with little or no sanitization. Let us take another example from real life to clarify the concept: the login functionality implemented in `view_account.cgi` takes two input parameters to authenticate users: a user ID (*num*) and password. If these values do not match any record in the database, the application will retrieve a message indicating such condition.

Inspection of the code in `view_account.cgi` shows that the SQL query authenticating the user goes as follows:

```
...
cu = mysql.prepareStatement(„SELECT FROM users WHERE
num=\'\" + (String)request.getParameter(„num“) + „\'
AND password LIKE \\'\" + (String)request.getParameter(„p
assword“) + „\'");
...
```

As can be seen, the aforementioned variables – provided by the user– are being used dynamically to construct the SQL statement. Thus, a malicious user could send a `userID` value such as `\ OR 'A' LIKE 'A` and a password like `123456`. That query should match every user having that password and authenticate the first of them: see Figure 10.

## A Word On Fixing SQL Injection Vulnerabilities

Most SQL injection vulnerabilities can be easily fixed by avoiding the use of dynamically constructed SQL queries and using parameterized queries instead. If it is not possible to use parameterized queries because the string appended is not a data type (for example, the name of the table in a CREATE SQL statement) it is still possible to sanitize the string to ensure that it cannot be used to trigger SQL injection vulnerabilities.

One option is to only allow alphanumeric characters. There are other characters that can be allowed (such as the underscore), but try to specifically avoid the following characters: double quotes, single quotes, semicolon, colon and dash. And remember that best practice is always restricting the allowed characters rather than filtering out specific bad ones – allow alphanumeric characters and discard everything else, rather than just filtering out single quotes.

## References

Here is a list on some textbooks that will give the grounds for this article

- Secure coding: principles & practices (Team LiB – O'Reilly)
- Writing Secure Code (Michael Howard – Microsoft Press)
- <http://www.springerlink.com/content/80wrewj7j1a716wb/>
- The .NET developers guide to Windows security
- The Databases hackers' handbook
- [http://www.databasesecurity.com/oracle/twp\\_security\\_db\\_vpd\\_10gr2.pdf](http://www.databasesecurity.com/oracle/twp_security_db_vpd_10gr2.pdf)

## Conclusion

Throughout this article we discussed a set of principles for secure application design and development, defined around the idea of thinking about the actors involved – users, devices, other applications, and etcetera– their roles and capabilities, and their interaction with the application itself. Then we introduced different vulnerabilities commonly found in real-life applications that might end up affecting different components: the server operating system in the case of *buffer overflows*, the user browser for *Cross-Site Scripting*, and the database in *SQL injection* attacks. While covering all of the know vectors would take several books – even this short analysis was based on many – it is a clever approach to analyze security in terms of which component is affected and who or what is affecting it. The knowledge of security threats can help preventing those specific threats from occur, whereas this way of thinking will lead architects and developers to be more cautious when building applications, and prevent most of the potential issues beforehand.

---

## JORGE LUIS ALVAREZ MEDINA

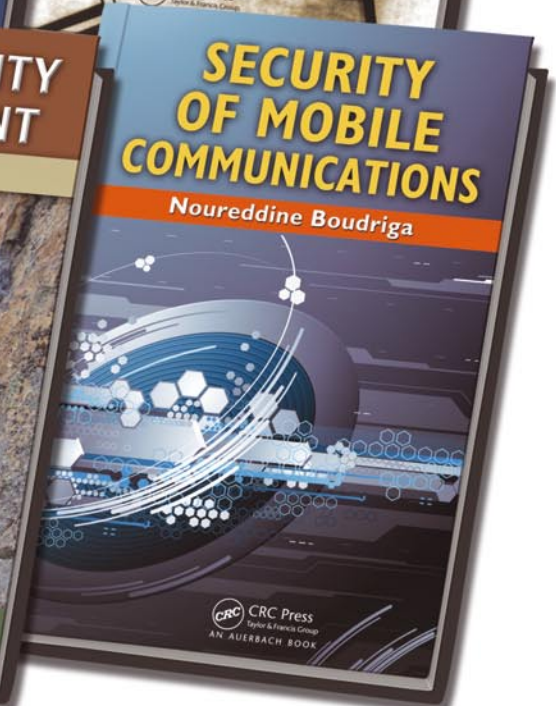
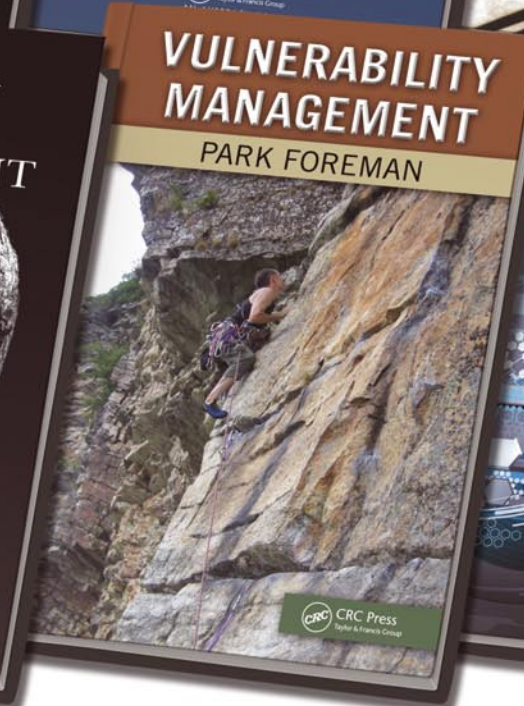
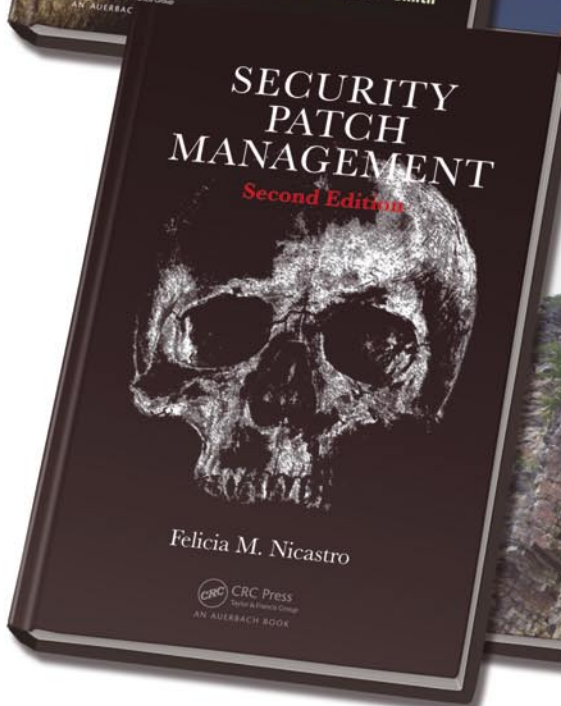
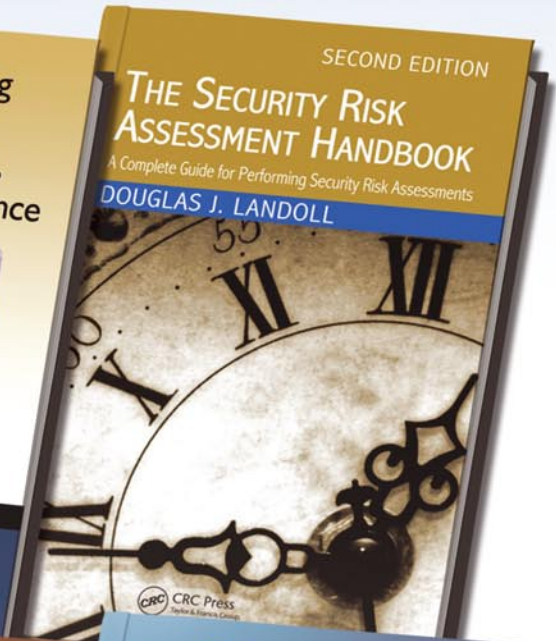
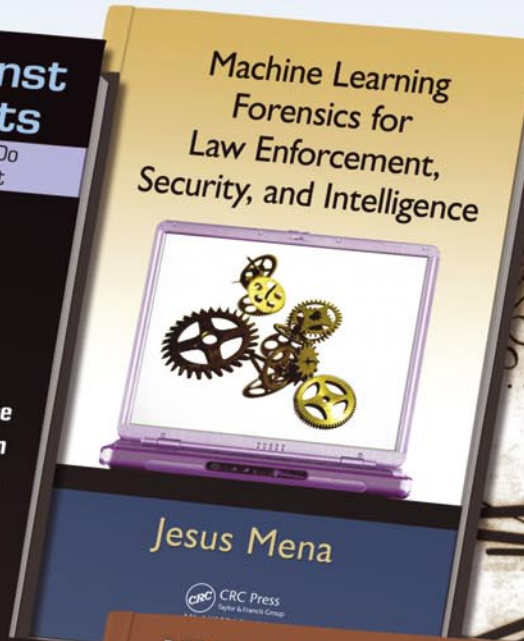
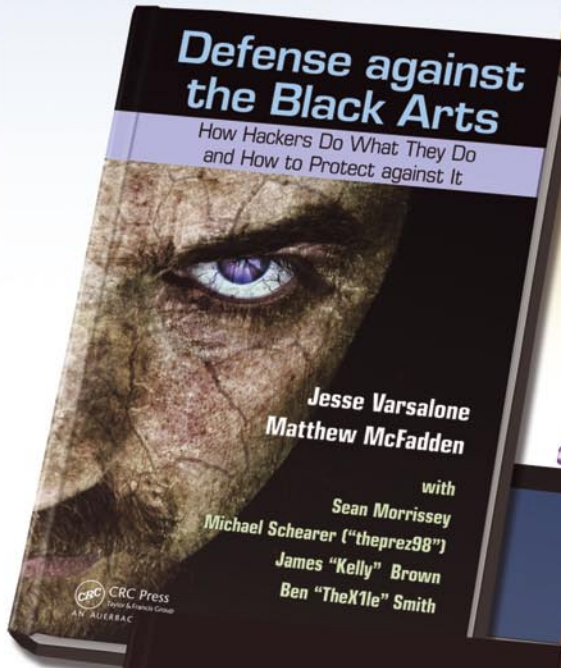
*Is an experienced security consultant and researcher. He works with leading firms in the security industry such as Buguroo, providing security services for several Fortune 500 companies. Some of his work as researcher was included in his presentation at Black Hat DC 2010, and in different security advisories for well-known applications.*





# Limited Time Offer

Secure Your System  
with these  
Critical Volumes



Enter promo code **510HA** at checkout to **SAVE 50%**

[www.crcpress.com](http://www.crcpress.com)



CRC Press  
Taylor & Francis Group

Offer expires 12/31/2011



# For My Eyes Only

Data at rest and company drive shares spell disaster. Learn how to protect yourself against your data be it your programs, scripts and allow automation to occur non-interactively without you having to type your password in because you don't want to save them within the execution file. This demonstration focuses on the Apple Mac platform but can be easily geared otherwise.

## What you will learn...

- You will learn to securely store your passwords at rest because they are uniquely bound to your machine.

## What you should know...

- You should know basic programming approaches for bash scripting.

Data is a marvelous thing; so easy to create but so difficult to keep track of and maintain. This marvelous thing is the very thing that can take companies down to their knees. All without anyone knowing until it is too late... A silent killer.

File shares are rampant with data, obsoleted, stagnated, and then... then sometimes useful. Most file shares be it on a filer, netapp or just a plain old windows xp admin share (c\$) over time accumulates access creep and where folks no longer maintain their data in a sane fashion; but more so just dump it there without regard for who has access to it or happens to come across it. Some folks even drag their entire *My Documents* folder directly into a public file share

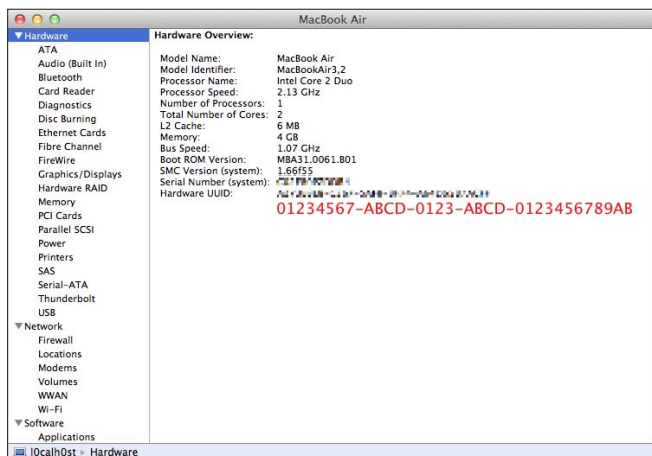


Figure 1. Apple Hardware UUID

without thinking about the nasty consequences to follow.

IT administrators (the so-called professionals) are often the worst of the bunch (*ala do as I say not as I do*). Usually overworked staff kept so busy they have no time to reflect, maintain or perform due attrition on lingering permissions and data those permissions allow. The best pieces of data are the ones they decided at one point or in another to store passwords to key systems in plaintext (*ala Notepad/Word/Excel*). Who has the time to

```

# this begin the main conditional flow
validate05

if [ ! $# -ne 2 ]; then
    operation=$1
    stringvar=$2
    if [ $operation == "-e" ]; then
        getUniqueID
        passEncrypt "$stringvar"
        exit $?
    fi
    if [ $operation == "-d" ]; then
        getUniqueID
        passDecrypt "$stringvar"
        exit $?
    fi
    if [ $operation == "-r" ]; then
        setDEMOUniqueID
        passEncrypt "$stringvar"
        exit $?
    fi
    if [ $operation == "-f" ]; then
        setDEMOUniqueID
        passDecrypt "$stringvar"
        exit $?
    fi
    usage
else
    usage
fi

#EOF
  
```

Figure 2. UUIDgenkey main

incorporate a password mechanism to maintain when it is far easier to copy and paste in a moment's notice? Even better are the scheduled maintenance scripts that are set to do things like reboot key systems when tolerances have been met... set in the fashion that within the script the passwords are plain as day – ripe for the picking.

This article will demonstrate how to simply and securely store your scripts so only you (the operator) can use them from your system no matter where you store the files. So even if someone finds your dirty laundry sheets swaying in the open wind, there is really nothing that can be done with them in a timely or useful manner (challenge accepted).

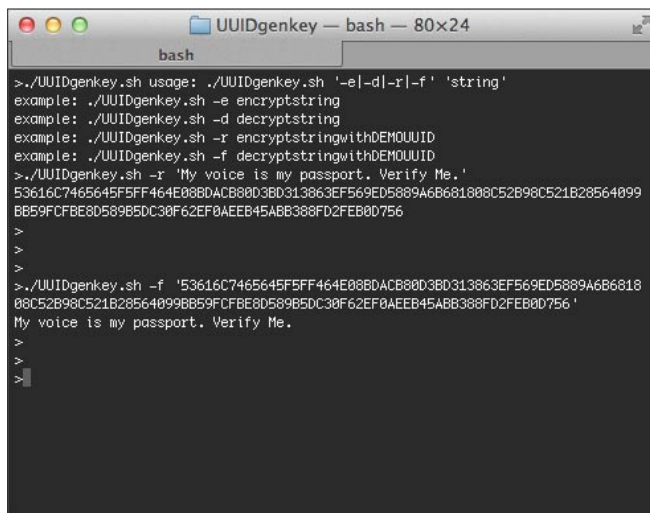
Apple machines such as a Macbook Air all have something called a Hardware UUID. UUID is short for *Universally Unique Identifier*. In theory it is accepted that no two machines will have the same UUID or be in a situation where they will be confused with one another (even across the vast reaches of the Internet).

Using the System Information App you get defaulted to immediately see the Hardware UUID on the lower right pane (Figure 1). Although pixelated in the figure a representation of this appears underneath in red. It is 36 characters in length and contains alphanumeric separated by dashes.

The Hardware UUID is also accessible using the `system_profiler` CLI tool (man `system_profiler`). The specific command to view what you see in System Information is :

```
system_profiler SPSHardwareDataTypee
```

For demonstrative purposes we'll be using a static DEMO UUID and storing in the variable labeled aptly `$myUUID`. This is done for two reasons; not to expose the true UUID of the test machine in question; but also to allow you to play with the scripts and bend them to



```

bash
./UUIDgenkey.sh usage: ./UUIDgenkey.sh [-e|-d|-r|-f] 'string'
example: ./UUIDgenkey.sh -e encryptstring
example: ./UUIDgenkey.sh -d decryptstring
example: ./UUIDgenkey.sh -r encryptstringwithDEMOUUID
example: ./UUIDgenkey.sh -f decryptstringwithDEMOUUID
./UUIDgenkey.sh -r 'My voice is my passport. Verify Me.'
53616C7465645F5FF464E08BDACB80D3BD313863EF569ED5889A6B681808C52B98C521B28564099
BB59FCFBE8D589B5DC30F62EF0AEEB45ABB388FD2FEB0D756
>
>
./UUIDgenkey.sh -f '53616C7465645F5FF464E08BDACB80D3BD313863EF569ED5889A6B681808C52B98C521B28564099BB59FCFBE8D589B5DC30F62EF0AEEB45ABB388FD2FEB0D756'
My voice is my passport. Verify Me.
>
>

```

Figure 3. UUIDgenkey usage and examples

your will until they are ready for the real UUID. It is certainly recommended not to use the DEMO UUID unless you specifically are testing something. Below we'll be testing the DEMO UUID against the real UUID to display what occurs when a password encrypted with a different UUID is encountered. This would be the case where someone found your UUID enabled script and tried to run it as-is.

UUIDgenkey.sh is a bash script created and tested on Mac OS X 10.7 Lion, however it works in other variations as well. The first function in the script checks to make sure the system is using Mac OS X as not to come up with platform compatibility issues without being tested first on an alternate platform. This function gets invoked as part of the validation process to proceed into the conditional workflow of the main script itself (Figure 2).

There are a total of five parameters it is built to recognize (Figure 3). 4 operational parameters and one string variable. Generally it could have used two operational parameters (encrypt,decrypt) but two more were added for the DEMO mode. They (-r & -f) are the same respectively (-e & -d) for the one exception that -r and -f are encrypting and decrypting against the DEMO UUID: 01234567-ABCD-0123-ABCD-0123456789AB. The real UUID is captured via the `getUniqueID` function where it makes a call to `system_profiler` and carves out the result into the variable `$myUUID` (Figure 4).

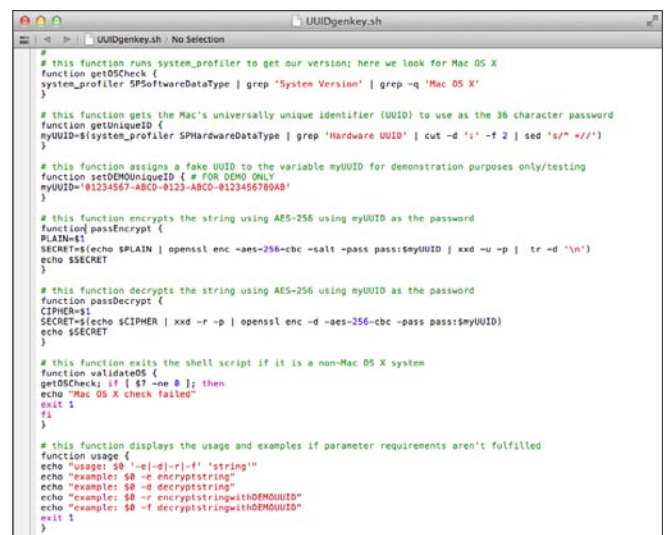
Below is how UUIDgenkey can be used in terminal or from another script:

```

./UUIDkey -e 'string' # use UUID to encrypt password
./UUIDkey -d 'string' # use UUID to decrypt password

./UUIDkey -r 'string' # use DEMO UUID to encrypt password
./UUIDkey -f 'string' # use DEMO UUID to decrypt password

```



```

UUIDgenkey.sh: No Selection
#
# this function runs system_profiler to get our version; here we look for Mac OS X
function getOSCheck {
system_profiler SPSHardwareDataTypee | grep 'System Version' | grep -q 'Mac OS X'
}

# this function gets the Mac's universally unique identifier (UUID) to use as the 36 character password
function getUniqueID {
myUUID=$(system_profiler SPSHardwareDataTypee | grep 'Hardware UUID' | cut -d '-' -f 2 | sed 's/" / /')
}

# this function assigns a fake UUID to the variable myUUID for demonstration purposes only/testing
function setDEMOUniqueID { # FOR DEMO ONLY
myUUID="01234567-ABCD-0123-ABCD-0123456789AB"
}

# this function encrypts the string using AES-256 using myUUID as the password
function passEncrypt {
PLAIN=$1
SECRET=$(echo $PLAIN | openssl enc -aes-256-cbc -salt -pass pass:$myUUID | xxd -u -p | tr -d '\n')
echo $SECRET
}

# this function decrypts the string using AES-256 using myUUID as the password
function passDecrypt {
CIPHER=$1
SECRET=$(echo $CIPHER | xxd -r -p | openssl enc -d -aes-256-cbc -pass pass:$myUUID)
echo $SECRET
}

# this function exits the shell script if it is a non-Mac OS X system
function validateOS {
getOSCheck; if [ $? -ne 0 ]; then
echo "Mac OS X check failed"
exit 1
fi
}

# this function displays the usage and examples if parameter requirements aren't fulfilled
function usage {
echo "usage: $0 [-e|-d|-r|-f] 'string'"
echo "example: $0 -e encryptstring"
echo "example: $0 -d decryptstring"
echo "example: $0 -r encryptstringwithDEMOUUID"
echo "example: $0 -f decryptstringwithDEMOUUID"
exit 1
}

```

Figure 4. UUIDgenkey functions

```

> ./UUIDgenkey.sh -d '53616c7465645f5ff464e08bdac80d3bd313863ef569ed5889a6b68180
8c52b98c521b28564099bb59fcf8e8d589b5dc30f62ef0aeeb45abb388fd2feb0d756'
bad decrypt
10896:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:/
SourceCache/openssl098/openssl098-41/src/crypto/evp/evp_enc.c:330:
?D?y?r?????H+np?k*?:????????K?
>

```

Figure 5. *bad decrypt*

For the actual encryption/decryption functions we are using openssl's version of a salted AES-256 result using the UUID as the password which is then converted to a hexadecimal string for portability.

*UUIDgenkey.sh* is then used to both encrypt and decrypt the generated string. If the UUID is not the same as expected there will be a returned error of a bad decryption (Figure 5); and the action will not succeed (as it would have on a system that just had the password in the clear). The return data can also be further modified not to display this but still understand the failure.

*UUIDgenkey.sh* is written to be used both standalone as well as from other scripts. See the example script labeled *example\_shutdown.sh* (Figure 6) which is written to use the credentials stored and calls *UUIDgenkey* to decrypt the password temporarily to execute a call. All from the operators system transparently.

*example\_shutdown.sh* stores the UUID bound password 53616c7465645f5ff464e08bdac80d3bd313863ef569ed5889a6b681808c521b28564099bb59fcf8e8d589b5dc30f62ef0aeeb45abb388fd2feb0d756 in a variable labeled `SECRETBIND` and at rest has no real issue of being found as discovery. Only when using the UUID (DEMO in this case) can we expose the variable called to store the decrypted value *My voice is my passport. Verify*

```

#!/bin/bash
# ./example_shutdown.sh
# Israel Torres hakin9@israelortorres.org
# Tue Aug 23 12:53:46 PDT 2011
# UUIDgenkey - For My Eyes Only"
# sample UUIDgenkey saved password script (shutdown)
#
# SECRETBIND contains the UUIDgenkey generated and stored prior using AES-salted-256
# ... only the user with the matching 36 character UUID can use this script transparently without further authentication safely
# ... the caveat being that their workstation is secured / and locked when the user is not using it.
SECRETBIND="53616c7465645f5ff464e08bdac80d3bd313863ef569ed5889a6b681808c521b28564099bb59fcf8e8d589b5dc30f62ef0aeeb45abb388fd2feb0d756"
# Using UUIDgenkey.sh the script temporarily saves the decrypted password into the variable ACCTPASS for use in the next command
ACCTPASS=$(./UUIDgenkey.sh -f $SECRETBIND)
# Using the net rpc shutdown command to shutdown a Windows machine from a Mac with valid credentials stored in SECRETBIND
net rpc shutdown -r -f -C "shutdown is imminent" -I 10.10.10.10 -U Administrator%ACCTPASS
# Changing the variable after use for the more paranoid at heart folks
ACCTPASS=$(for i in {1..100}; do echo -n $RANDOM; done)
#EOF

```

Figure 6. *example\_shutdown bash script*

*Me*, labeled `$ACCTPASS`. After executing the `net rpc shutdown` command using the decrypted authorized credentials we then again quickly modify `$ACCTPASS` (for the paranoid folks out there) using a large buffer from a chain of `$RANDOM` values.

As demonstrated the script *example\_shutdown.sh* executes transparently on your specific system; but does not execute correctly on a different system. The only real burden is to use *UUIDgenkey* to generate different UUID keys for each of your machines if you need to run the script from all machines. This is simply done with a script if you have all the UUIDs saved in a text file and just loop *UUIDgenkey* through. Naturally keeping a list of UUIDs isn't recommended with this approach as an attacker may be able to put together what is happening and recreate and come to learn the decrypted passphrase from the UUID.

### Note

Both source files *UUIDgenkey.sh* and *example\_shutdown.sh* have been made available for you to download and modify. If you cannot locate them please do not hesitate to contact me for a download link (information below).

From what you've learned you can now create various bash files/AppleScripts that perhaps you would have hesitated in adding password information. (FTP scripts, curl scripts) Again, I stress that only do so if you feel it is adequate and valid for the situation at hand. Most importantly keep track and change your authentication values on a regular basis and synchronize accordingly. Scripts nowadays are so versatile that it is a shame not to take advantage of them especially when it involves having to interact with a running script (before it times out) to run something you could have alternatively used *UUIDgenkey* to safely automate the process entirely.

Using alternative serialization may also make due as long as you stay consistent and the data is readily available without too much tweaking. For example you can also incorporate your username with the UUID and the password by concatenating your username to the UUID prior to the encryption as long as you match it upon the decryption (read queried system variables that are unique to your machine in one way or another – even checking the serial number of a USB key you just inserted).

The key points to validate this type of system is to make sure whichever data you are using UUID, Serial Number, Username,



## Web Links and References

- [http://en.wikipedia.org/wiki/Universally\\_unique\\_identifier](http://en.wikipedia.org/wiki/Universally_unique_identifier)
- <http://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man3/uuid.3.html>
- <http://en.wikipedia.org/wiki/Encryption>

## Notes

All source code created and tested on:  
Mac OS X 10.7.1 11B26  
Intel Core 2 Duo 2.13 GHz 4 GB  
Darwin Kernel Version 11.1.0  
GNU bash, version 3.2.48(1)-release

## Got More Time Than Money?

Try this month's crypto challenge:  
<http://hakin9.israeltorres.org>

specialized token that you keep as many components secret else fall victim to someone that takes this information and recreates it to imminently gain the reversed password you are trying to protect. This isn't the same as the concept of security through obscurity as you are using hooked values (hard and soft tokens) that naturally only you know/have and the attacker either needs to make effort to know them (and spoof them) or even more considerable effort to brute force them.

## Conclusion

Data at rest is a most dangerous target because whilst it is at rest it is vulnerable to being copied without anyone the wiser. Further security would be to have the hard disk (or SSD) set to be wholly encrypted. So, even if the drive is pulled out temporarily without your knowing, the attacker doesn't have anything of value (well unless they know what you know, and then well you are doomed).

This is why it is so important to not put your data on file shares. You don't know who is watching them and if they are even being maintained. When is the last time you've had your document sharing system audited for access creep and folder sprawl? If you don't know who does?

If you are going to put them on these open shares you can sleep peacefully knowing that no matter where they are they will only work on your system from your system.

## ISRAEL TORRES

*Israel Torres is a hacker at large with interests in the hacking realm.*

[hakin9@israeltorres.org](mailto:hakin9@israeltorres.org)

[http://twitter.com/israel\\_torres](http://twitter.com/israel_torres)

<https://plus.google.com/102921309581624765133/posts>



[ GEEKED AT BIRTH ]

You can talk the talk.  
Can you walk the walk?

[ IT'S IN YOUR GENETICS ]

LEARN:

Advancing Computer Science	Network Security
Artificial Life Programming	Open Source Technologies
Digital Media	Robotics and Embedded Systems
Digital Video	Serious Games and Simulation
Enterprise Software Development	Strategic Technology Development
Game Art and Animation	Technology Forensics
Game Design	Technology Product Design
Game Programming	Technology Studies
Human-Computer Interaction	Virtual Modeling and Design
Network Engineering	Web and Social Media Technologies

[www.uat.edu](http://www.uat.edu) > 877.UAT.GEEK

Please see [www.uat.edu/fastfacts](http://www.uat.edu/fastfacts) for the latest information about degree program performance, placement and costs.



# Secure Coding PHP

It can be said that software is only as good as its code or as good as the developer who wrote that code. Yet if we used this adage to compare current web based software, we are in need of some major retrofits to the software we entrust our personal data to.

## What you will learn...

- How to write secure code in PHP
- How to validate user input
- Encryption techniques
- Other counter measures

## What you should know...

- How to write code using PHP

The recent cyber attacks on BART – the San Francisco Bay Area’s rapid transit system – only demonstrates the need for better and more secure software especially when personal and private information is at stake. As cyber attacks only seem to be growing in number, we have to start to focus more on secure coding as we try to walk the thinning line that is security and usability. With this in mind, we will discuss some of the techniques one can use to write more secure PHP code including user input verification and data encryption. And as we move through these techniques, we will see that building a more robust and secure web application will not only increase the user’s confidence in their information staying secure, but it will also avoid the software becoming the target of a cyber attack.

## Securing Web Forms And Checking User Data

Web forms are one of the primary ways in which we interact with web based applications. As a result, we are entrusting our personal data to the security put in place by the web developer – if they even added anything in the first place. For example, if we were to have a web form through which users could add comments to a web site or the like, this would be a prime opportunity for a cross-site scripting vulnerability mainly because such pages allow for the inclusion of HTML tags.

Looking at Listing 1B, we notice that there are no controls in place to prevent the inclusion of HTML code

**Listing 1A.** *Commenting page – HTMLlang=python*

```
<html>
  <head>
    <title>Comment Page</title>
  </head>
  <body>
    <form action="comment_action.php"
          method="POST">
      <table borders="0">
        <tr>
          <td>Add a comment:<textarea
            name="comment" cols="55"
            rows="5"></textarea></td>
        </tr>
        <tr><td><input type="submit"
          name="submitBtn" value="Post"
          /></td></tr>
        <tr><td><input type="reset"
          name="clearBtn" value="Clear"
          /></td></tr>
      </table>
    </form>
  </body>
</html>
```

through the web form – as it can be seen in Figure 1A. Finally, looking at Figure 1B it's quite obvious that our HTML made it through and at this point we would know that we could pass more malicious code through the form.

Having seen how easy it is to create but also search for a cross-site scripting vulnerability, there are a number of methods to avoid such problems from even occurring. Specifically within the PHP language, there are multiple functions which can help us, these are: `escapeshellarg()`, `escapeshellcmd()`, `htmlspecialchars()`, and `strip_tags()`. As a quick overview, the first two functions ensure that shell

#### Listing 1B. Commenting page – PHP

```
<?php
//Variable declarations
$commentPassthrough = "";
$commentHE = "";
$commentST = "";

//Get values from form
$commentPassthrough = $_POST["comment"];
$commentHE = htmlspecialchars($_POST["comment"]);
$commentST = strip_tags($_POST["comment"]);
$commentAcceptedTags = strip_tags($_
    POST["comment"], "<b></b><i></i><br>");

//Print to the screen
echo "---No security measures---<br />";
echo $commentPassthrough . "<br />";

echo "<br />";

echo "---htmlspecialchars()---<br />";
echo $commentHE . "<br />";

echo "<br />";

echo "---strip_tags()---<br />";
echo $commentST . "<br />";

echo "---strip_tags() with acceptable tags---<br
    />";
echo $commentAcceptedTags . "<br />";

echo "<br />";
?>
```

commands can't be passed through and subsequently executed using another function such as `exec()` or `system()`, though they don't prevent cross-site scripting. The other two functions – `htmlspecialchars()` and `strip_tags()` ensure that HTML can't be passed through a web form unless acceptable tags are specified.

Because not many applications are executing system level commands that often on the web, we won't delve into that detail, however, we will offer an overview as to how the `escapeshellarg()` and `escapeshellcmd()` functions work. As was just mentioned, the `exec()` and `system()` functions would allow an user user to execute system level commands from a PHP application. If this is the case and there is no security in place, the user could quite literally erase all of the files within a directory for example. While this is not really a *hack*, such functions such as `escapeshellarg()` and `escapeshellcmd()` could create a potential problem within an.

Using the `escapeshellarg()` function in a web form, it would essentially ensure that appending extra arguments to the end of an *expected command* would not be possible. Essentially the function creates a single argument rather than allowing multiple arguments to be passed through. The `escapeshellcmd()` is similar in functionality to the `escapeshellarg()` command except that it escapes the meta-characters that a shell might



Figure 1A. Passing HTML through form

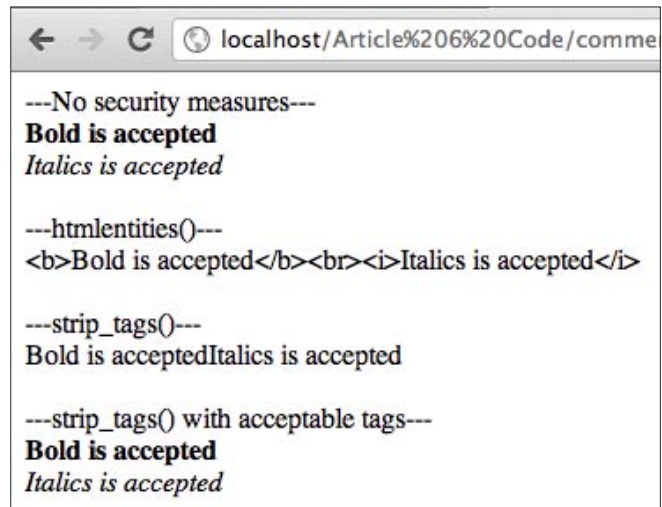


Figure 1B. The results

**Listing 2A. Email page – HTML**

```
<html>
  <head>
    <title>Email Page</title>
  </head>
  <body>
    <form action="email_action.php"
      method="POST">
      <table borders="0">
        <tr><td>Email: <input type="text"
          name="email" size="35" /></td></tr>
        <tr><td><input type="submit"
          name="submitBtn" value="Post"
        /></td></tr>
        <tr><td><input type="reset"
          name="clearBtn" value="Clear"
        /></td></tr>
      </table>
    </form>
  </body>
</html>
```

**Listing 2B. Email page – PHP**

```
<?php
//Variable declarations
$emailPassThrough = "";
$emailFiltered = "";

//Get values from form
$emailPassThrough = $_POST["email"];
$emailFiltered = strip_tags($_POST["email"]);

//Print to the screen
echo "---No security measures---<br />";
echo $emailPassThrough . "<br />";

echo "<br />";

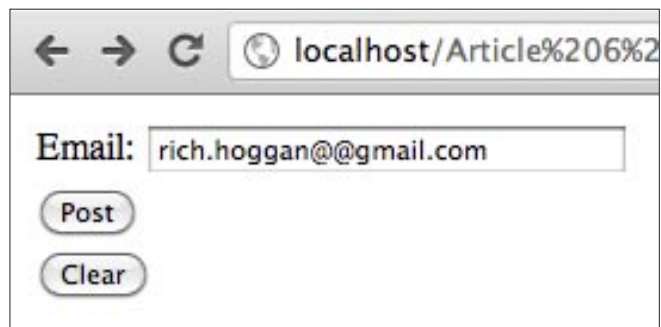
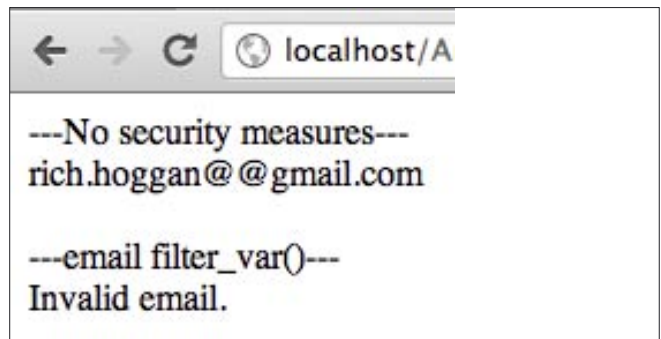
echo "---email filter_var()---<br />";

{
  echo "Valid email.<br />";
  echo "<br />";
}
else
{
  echo "Invalid email.<br />";
  echo "<br />";
}
?>
```

be expecting. In this case, it essentially escapes program names rather than erroneous arguments.

Let's now have a closer look at the `htmlspecialchars()` and `strip_tags()` functions. Depending on what your purpose is, you can either remove the meaning of the HTML tags to the browser or you can eliminate them from the user's input completely. Looking to Listing 1B once more, you will notice that we use the `htmlspecialchars()` and `strip_tags()` functions when we are getting the form data using the POST method. As such, the `htmlspecialchars()` function is the one to use if you want to remove the meaning of all HTML tags to the browser. Doing so ensures that a cross-site scripting vulnerability doesn't exist within your web form and also ensures that while an attacker can pass HTML tags through a web form, the browser doesn't interpret what it sees as an HTML tag.

Similarly, the `strip_tags()` function offers up the similar functionality in what it accepts form data and instead of removing the meaning of the tags from the attacker's input it simply removes the tags. This way if you happen to be storing input from the form in a database, the tags won't be present. Something to keep in mind though is the fact that the `strip_tags()` function allows you to specify acceptable tags to be passed through the web form. For example if the user wished to add his or her own HTML tags to the input – usually in a blog entry or the like – they are able to do so. Looking at Listing 1B you will notice that we have another variable that allows us to pass accepted HTML tags. In order to specify acceptable tags we list each of the tags as the second argument to the `strip_`

**Figure 2A. Entering an email address****Figure 2B. The results**

`tags()` function. In this case, we are accepting bold, italics, and break – three of the more commonly used tags.

We now move into a different type of validation where we are checking the user's input rather than escaping or stripping it. As a simple example, we are validating the user's email address using the `filter_var()` function. This function offers up a lot of validation options, all using one function and different validation flags.

You will notice in Listing 2B that we are still using the `strip_tags()` function because the user could still attempt malicious actions with any of the form's elements. But once we have stripped away any potential HTML tags from the input we then check that the email matches a typical email address. As can be seen in Figure 2B, when we aren't validating the email address, the user could enter pretty much anything they want but with the `filter_var()` function we are able to ensure the email address is valid.

Aside from simply validating user input, it's also possible to sanitize user input using the `filter_var()` function's sanitization flags. Doing so gives us the ability to sanitize things such as integer values, floating point values, and strings amongst others.

#### Listing 3A. Login page – HTML

```
<html>
  <head>
    <title>Login Page</title>
  </head>
  <body>
    <form action="login_action.php"
          method="POST">
      <table borders="0">
        <tr><td>Username: <input type="text"
                               name="username" size="35" /></td></tr>
        <tr><td>Password: <input
                               type="password" name="password"
                               size="35" /></td></tr>
        <tr><td><input type="submit"
                               name="submitBtn" value="Login"
                               /></td></tr>
        <tr><td><input type="reset"
                               name="clearBtn" value="Clear"
                               /></td></tr>
      </table>
    </form>
  </body>
</html>
```

## Data Encryption: Keeping Private Data Private

So far we have covered the process of validating and sanitizing user input. Another technique for creating secure code is data encryption. By using encryption, we are able to ensure that even if an attacker were able to access login data through a cross-site scripting attack, they would end up with an encrypted hash value rather than the actual password. Specifically, within PHP, there are a couple of options available to us. The first is to use hashing algorithms such as `md5()` and `sha1()` and the second is to use MCrypt.

What makes using hashing algorithms worthwhile, such as the ones we just listed, is the fact that they are considered one-time pads in that the algorithms can't be decrypted. This means that it will end up being a little more difficult for an attacker to figure out what was originally encrypted.

#### Listing 3B. Login page – PHP

```
<?php
//Variable declarations
$username = "";
$passwordPassThrough = "";
$passwordEncryptedMD5 = "";
$passwordEncryptedSHA1 = "";

//Get values from form

$passwordPassThrough = strip_tags($_
    POST["password"]);
$passwordEncryptedMD5 = strip_tags(md5($_
    POST["password"]));
$passwordEncryptedSHA1 = strip_tags(sha1($_
    POST["password"]));

echo "---No Security Measures---<br />";
echo $passwordPassThrough . "<br />";

echo "<br />";

echo "---Encrypted using md5()---<br />";
echo $passwordEncryptedMD5 . "<br />";

echo "<br />";

echo "---Encrypted using sha1()---<br />";
echo $passwordEncryptedSHA1 . "<br />";

echo "<br />";
?>
```



Figure 3A. Login page – Passing input

Figure 3B. Login page – The results

Before we get into how to use these functions, it should first be pointed out that using such algorithms does you no good unless you are using a server that is SSL capable. This is because the data we are sending to the server is obviously clear text prior to it being encrypted.

Looking at Listing 3B and Figure 3B, we have implemented the `md5()` as well as the `sha1()` hashing algorithms. You will also notice that once the password is encrypted it ends up looking like a string of characters and numbers – otherwise known as a hash value.

As was just clarified, the encryption algorithms we just discussed don't allow for the decryption of cipher text. If you wish to secure your PHP scripts using encryption but want the ability to decrypt encrypted data, the `MCrypt` package (<http://mcrypt.sourceforge.net>) is your best bet. Using this package, you are able to encrypt and decrypt using a multitude of encryption/decryption ciphers – Blowfish, DES, and 3DES for example. It should be noted though, that you can't use this package until it has been downloaded and installed – obviously.

### Other Options

Some of the less technical options we have to help secure our code are to keep our technical details out of view of a potential attacker. This means not displaying information about the version of Apache or PHP you are using. To do this for Apache, we use the `ServerSignature` and `ServerTokens` configuration directives. Because so much information is displayed including the version number, server, and port number, the `ServerSignature`

should be turned off completely. In doing so the server's entire signature won't be displayed, but if you must display this information, the `ServerTokens` configuration directive allows you to specify certain levels of display all the way from the entire signature to simply `Apache Server`. Other display options include Major, Minimal, Minor, and OS.

When it comes to hiding PHP details, the `expose_php` configuration directive allows you to determine whether or not the version number is displayed in the server signature. Something to keep in mind is that this configuration directive works with the `ServerSignature` and `ServerTokens` directives. For example, if the `ServerSignature` directive is enabled and the `expose_php` directive is enabled, PHP's version number will be displayed in the server signature. However, if the `expose_php` directive is not enabled, PHP's version number will not be displayed in the server signature. Another important detail to remove is any calls to the `phpinfo()` function. Not allowing this function to be called means that an armada of information will not be available to a potential attacker – thus avoiding a potential cyber attack or displaying any vulnerabilities that might have been ascertained by the provided information.

### Conclusions

Secure coding in PHP can be likened to a tool in the toolbox. This is one option in combating the problem of cyber attacks and private data being accessed. Similarly, ensuring users to change their passwords on an annual basis as well as choosing strong passwords from the get go also helps in minimizing the risk to sensitive systems and data. For the most part, secure coding comes down to understanding your development platform and the application you wish to create. Once you have an idea of what needs to be created you can then use secure coding principles as you write the software's code. Doing so will not only create a more robust and secure web application, but will also keep the attackers at bay.

---

### RICH HOGGAN

*Rich Hoggan is currently a third year Computer Science major at the University of San Francisco and plans to specialize in Cyber Security and Information Assurance. When not writing he is involved in the electronic music scene and creates photographic art using the Processing programming language.*

# iswec 2012

Infosecurity World Exhibition & Conference



21 - 22 March 2012  
Putra World Trade Centre, Kuala Lumpur

Incorporating:



Concurrently held with:



**A Central Meeting Place for Information Security Industry**

Organized by

Supported by

Official Contractor

Official Hotel

Official Freight Forwarder



Media Partners



HAKING



SECURITY4INDIA



For more info, contact us at

Malaysia Office ☎ +603-2600 6000

Shanghai Office ☎ +86-21-5172 0000

✉ secretariat@infosecurityworld.net

or visit us at 🌐 [www.infosecurityworld.net](http://www.infosecurityworld.net)

# Secure Coding in the Database

Information systems are not islands. Either data is manually entered, or, as is more commonly the case, interchanged with other systems.

## What you will learn...

- Creating automatic audit trails for critical database tables
- Creating processes to guard against and recover from bad data
- Building a lightweight process for rapid data recovery that avoids using complex, time-consuming database backup tools

## What you should know...

- Basic understanding of creating tables in a database

Some systems are very tightly integrated: a database transaction committed in one system becomes available in another almost immediately. Other systems are more loosely coupled and synchronize data on a scheduled basis. Some partners in the interchange do an outstanding job of vetting their data and making sure that the data feeds

are clean. But what do you do when a data supplier comes under attack, the data becomes vandalized, or it is rendered unavailable? There are techniques to secure data, to make sure it isn't contaminated with errors, and to land on your feet without resorting to lengthy database recoveries even if corrupted data is loaded.

**Table 1.** Column information on the staging table for employee data, including name, data type, null value behavior, and the column sequence order

COLUMN_NAME	DATA_TYPE	NULL ALLOWED	DATA DEFAULT	COL_NO
DIVISION	VARCHAR2(60 CHAR)	Yes	null	1
EMPLOYEEENAME	VARCHAR2(60 CHAR)	Yes	null	2
DISPLAYNAME	VARCHAR2(150 CHAR)	Yes	null	3
TITLE	VARCHAR2(60 CHAR)	Yes	null	4
DEPARTMENT	VARCHAR2(60 CHAR)	Yes	null	5
PHYSICALADDRESS	VARCHAR2(60 CHAR)	Yes	null	6
MAILINGADDRESS	VARCHAR2(200 CHAR)	Yes	null	7
POSTALCODE	VARCHAR2(60 CHAR)	Yes	null	8
EMPLOYEEID	VARCHAR2(60 CHAR)	Yes	null	9
EMPLOYEEENUNBER	VARCHAR2(60 CHAR)	Yes	null	10
NICKNAME	VARCHAR2(60 CHAR)	Yes	null	11
TELEPHONENUMBER	VARCHAR2(60 CHAR)	Yes	null	12
ROOMNUMBER	VARCHAR2(60 CHAR)	Yes	null	13
STATEPROVINCENAME	VARCHAR2(60 CHAR)	Yes	null	14
MAIL	VARCHAR2(150 CHAR)	Yes	null	15
FACSIMILETELEPHONENUMBER	VARCHAR2(60 CHAR)	Yes	null	16

Many good software engineering techniques also happen to ensure secure coding by anticipating events that might never come to pass. In this article, techniques for 1) dealing with data submission from untrusted sources, 2) sanitizing data, 3) keeping an audit trail, 4) comprehending differences, and 5) programmatic reversion of unwanted changes will help protect your system from compromise.

## Dealing With Data Submission From Untrusted Sources

Given the internetworking of database systems and the rise of computing power to quickly break authentication systems, all data suppliers are subject to compromise. They can suddenly turn from trusted to untrusted sources. To regard all data sources as potentially untrustworthy promotes the most secure approach.

A primary strategy, therefore, is to isolate incoming data from the main data in the system until certain validation requirements have been met. This strategy works well with loosely coupled systems that rendezvous on a daily basis.

When interacting with systems that provide a flat file output, storing the file in a *staging area* prior to uploading it into the database provides the opportunity to scan it for defects. Any problems found in the incoming data may provide sufficient cause to reject its submission without affecting the database.

## Sanitizing Data In Staging Tables

Similarly, *staging tables* in the database provide an area to load data and perform additional data checks that relate to business rules in the database. A staging table is either identical to or a close match with the main

### Listing 1. Cursor code in Oracle PL/SQL

```

1.  CURSOR changing_records_cur IS
2.      SELECT MIN(tablename) as tablename,  division, employeename, displayname, title,
3.      department, physicaladdress, mailingaddress,  postalcode,
4.      employeeid,  employeenumbr, givenname,  telephonenumber, roomnumber,
5.      stateprovincename,  mail, facsimiletelephonenumber
6.  FROM
7.      ( /* DISTINCT operator eliminates any chance of duplicates in external table */
8.      SELECT DISTINCT 'NEWROW' AS tablename,  division, employeename, displayname,
9.
10.     department, physicaladdress, mailingaddress,  postalcode,
11.     employeeid,  employeenumbr, givenname,  telephonenumber, roomnumber,
12.     stateprovincename,  mail, facsimiletelephonenumber
13.     FROM shared.fda_maildir_ext
14.     UNION ALL
15.     SELECT 'OLDROW' AS tablename,  division, employeename, displayname, title,
16.     department, physicaladdress, mailingaddress,  postalcode,
17.     employeeid,  employeenumbr, givenname,  telephonenumber, roomnumber,
18.     stateprovincename,  mail, facsimiletelephonenumber
19.     FROM shared.fda_maildir
20.     ) tmp
21.  GROUP BY
22.     division, employeename, displayname, title,
23.     department, physicaladdress, mailingaddress,  postalcode,
24.     employeeid,  employeenumbr, givenname,  telephonenumber, roomnumber,
25.     stateprovincename,  mail, facsimiletelephonenumber
26.     /* COUNT = 1 means record exists in either old or new tables-- but not both */
27.     /* COUNT = 2 means record is found in both old and new tables */
28.  HAVING COUNT(*) = 1
29.  ORDER BY
30.     division, employeename, displayname, title,
31.     department, physicaladdress, mailingaddress,  postalcode,
32.     employeeid,  employeenumbr, givenname,  telephonenumber, roomnumber,
33.     stateprovincename,  mail, facsimiletelephonenumber ;

```



**Listing 2.** Code fragment of procedure to update changes in the Employee\_Main table

```

1. BEGIN
2. --
3. -- initialize counters
4. -- interrogate environment and set global variables in package
5. --
6. i:=0;
7. iold:=0;
8. inew:=0;
9. g_updating_user := SYS_CONTEXT ('USERENV', 'SESSION_USER');
10. g_updating_time := SYSDATE;
11. g_employee_state := 'UPDATING';
12.
13. dbms_output.enable(NULL);
14. FOR rec IN changing_records_cur LOOP
15.
16.     CASE WHEN rec.tablename = 'OLDROW' THEN
17.         DELETE FROM employee_main
18.         WHERE
19.             division = rec.division AND
20.             employeename = rec.employeename AND
21.             displayname = rec.displayname AND
22.
23.             department = rec.department AND
24.             physicaladdress = rec.physicaladdress AND
25.             mailingaddress= rec.mailingaddress AND
26.             postalcode = rec.postalcode AND
27.             employeeid = rec.employeeid AND
28.             employeenumber = rec.employeenumber AND
29.             givenname = rec.givenname AND
30.             telephonenumber = rec.telephonenumber AND
31.             roomnumber = rec.roomnumber AND
32.             stateprovincename = rec.stateprovincename AND
33.             mail = rec.mail AND
34.             facsimiletelephonenumber = rec.facsimiletelephonenumber ;
35.         /*
36.         ** For each row deleted, a trigger creates a record in the history table.
37.         ** See DEL_EMPLOYEE_MAIN_TRG for details.
38.         */
39.         iold :=iold +1;
40.     ELSE -- case 'NEWROW'
41.         INSERT INTO employee_main VALUES (
42.             rec.division, rec.employeename, rec.displayname, rec.title, rec.department, rec.physicaladdress,
43.             rec.mailingaddress, rec.postalcode, rec.employeeid, rec.employeenumber, rec.givenname, rec.telephonenumber,
44.             rec.roomnumber, rec.stateprovincename, rec.mail, rec.facsimiletelephonenumber,
45.             -- audit columns
46.             g_updating_time, g_updating_user );
47.         inew :=inew +1;
48.     END CASE;
49.     i:=i+1;
50.     dbms_output.put_line(TO_CHAR(i)||':'||TRIM(rec.tablename)||':'||TRIM(rec.displayname)||':'|| rec.mailingaddress);
51. END LOOP;

```



# CYBER DEFENCE SUMMIT مؤتمر الأمن السيبراني

SEPTEMBER 20<sup>TH</sup> - 21<sup>ST</sup> 2011  
ABU DHABI, UAE  
[WWW.CYBERDEFENCESUMMIT.COM](http://WWW.CYBERDEFENCESUMMIT.COM)

## ATTENTION: CYBER SECURITY SOLUTION PROVIDERS MEET UP TO 120 ORGANISATIONS SEEKING PROTECTION

TELECOM & IT SERIES



### PLATINUM SPONSOR



### GOLD SPONSORS



### BRONZE SPONSORS



### ASSOCIATION PARTNERS



### MEDIA PARTNERS



For more details on participation, kindly contact: **Ali Rana**, Email: [register@cyberdefencesummit.com](mailto:register@cyberdefencesummit.com) | Tel: +971 4 367 1376

table in the database that holds incoming data prior to merging valid rows into the main table.

Consider the following staging table's columns in Table 1.

This data structure can accept almost any character input. There's no attempt to convert character input into numeric values nor require a column to have any value: it can have any kind of data as long as it does not overflow its generous allotment of characters.

Some database vendors make the load process easier with an external tables interface that combines data movement tools with an instantiated view of the flat file as if it were a table. (Refer to Oracle's online technical documentation to see how to create an external table having DataPump or SQL\*Loader semantics with read or write capabilities. [http://download.oracle.com/docs/cd/B19306\\_01/server.102/b14215/et\\_concepts.htm](http://download.oracle.com/docs/cd/B19306_01/server.102/b14215/et_concepts.htm))

Even without fancy external table components in the database, the ability to further manipulate the staging tables greatly helps in assessing and sanitizing data.

### Understanding Change With Audit Trails

Another good software engineering practice that promotes secure coding is building audit trails into applications. With careful preservation of former values in history tables, it's possible to revert unintended or unwanted changes.

A data supplier might have problems producing an output file, particularly if there were an attack on the data supplier's operation center. For example, if the data content had been compromised, or the ETL process had failed (producing a zero byte file for example), then the data center might experience a denial of service until the last good state was retrieved from a backup.

A denial of service situation can be corrected by keeping a history of recently changed records, whether by update, insert, or deletion of records, and building tools to re-create the last known good state. What's simple and powerful about this approach is that constructing a history table is easy. Adding fields such as `UPDATED_DATE` or `UPDATED_BY` to the tables helps in tracking the persistence and change of the information in the system. Complacency and the lack of awareness of good data vs. bad data and being unable to identify a point in time when something goes wrong is the kind of policy that a hacker could exploit. This kind of metadata is important to data management even if the database administrator is not examining computational forensics post-attack.

### Secure Versus Insecure Database Coding

In the following scenario, the business rule calls for dropping all the rows in the main table and replacing it with the data in the flat file.

**Listing 3.** Trigger code on the `Employee_Main` table that executes before deleting a row

```
TRIGGER CODE
1. TRIGGER DEL_EMPLOYEE_MAIN_TRG
2. BEFORE DELETE
3. ON EMPLOYEE_MAIN_TRG
4. FOR EACH ROW
5. BEGIN
6. IF (utl_employee_main.g_employee_state='UPDATING') THEN
7.     INSERT INTO employee_main_hist VALUES (
8.
9.         :old.division, :old.employeeename, :old.displayname, :old.title, :old.department,
10.         :old.physicaladdress, :old.mailingaddress, :old.postalcode, :old.employeeid,
11.         :old.employeeenumber, :old.nickname, :old.telephonenumber,
12.         :old.roomnumber, :old.stateprovincename, :old.mail, :old.facsimiletelephonenumber,
13.
14.         utl_employee_main.g_updating_time,
15.         utl_employee_main.g_updating_user );
16. ELSIF
17. NOT (utl_employee_main.g_employee_state='REVERTING') THEN
18.     RAISE_APPLICATION_ERROR(-20333,'Sorry -- no deletes allowed to this table.');
```

**Listing 4a.** PL/SQL code to delete records from main table as first step of undo procedure

```

52. PROCEDURE UNDO_CHANGES ( p_sqlcode OUT NUMBER, p_sqlerrm OUT VARCHAR2)
53. IS
54. /* Algorithm:
55. ** Use the most recent updated_date from the history table
56. */
57.   undo_user   EMPLOYEE_MAIN.UPDATED_BY%TYPE
58.             := SYS_CONTEXT ('USERENV', 'SESSION_USER');
59.   undo_time   EMPLOYEE_MAIN.UPDATED_DATE%TYPE ;
60.   i PLS_INTEGER;
61. /*
62. ** define collection to hold restored records
63. */
64.   TYPE emain_tabl_type IS TABLE OF EMPLOYEE_MAIN%ROWTYPE;
65.   restored_emain      emain_tabl_type;
66.   v_module_name      VARCHAR2(30) := 'REVERT_CHANGES';
67.   v_code NUMBER := 0;
68.   v_errm VARCHAR2(4000) := 'OK';
69. BEGIN
70. -- set initial parameters
71.   i:=0;
72. --
73. -- global package variable to indicate type of change to EMPLOYEE_MAIN
74. --
75.   g_employee_state := 'REVERTING';
76.
77.   i:=0;
78.   undo_time := get_last_change_dt; -- call to function inside this package.
79.   IF undo_time IS NOT NULL THEN
80.     dbms_output.enable(NULL);
81.     BEGIN
82.       DELETE FROM employee_main
83.       WHERE updated_date = undo_time;
84.
85.       /*
86.       ** For each row deleted, a trigger fires, however it does not create
87.       ** a record in the history table while g_employee_state = 'REVERTING'.
88.       ** See delete trigger for details.
89.       */
90.
91.     EXCEPTION
92.       WHEN NO_DATA_FOUND THEN
93.         v_code := 0;
94.         v_errm := v_module_name||' - NO DATA FOUND to delete. Continuing ';
95.       WHEN OTHERS THEN
96.         v_code := sqlcode;
97.         v_errm := sqlerrm;
98.         g_employee_state := NULL;
99.         RAISE;
100. END;

```



## Insecure Coding Approach

- Determine that flat file exists.
- Determine that database table exists.
- Delete all rows in main table.
- Read and insert all rows from the flat file and insert them into the main table.

## Secure Coding Approach

- Load data into staging table.
- Perform security checks on data.
- Screen out duplicates.
- Whenever there is a difference between existing content and new content read from the staging tables, then the correct action is to move the existing record to an audit or history table and permit the new record to be added.
- Prevent table manipulation unless through packages and stored procedures.
- Build process to remove the application of bad data in a single call.
- Build process to preview changes before committing them to the database.

## Detecting Differences And Handling With Secure Coding

Database cursors, a feature of PostgreSQL, SQL Server, DB2, Oracle database management systems (DBMS), help build and compile into code queries that can be used to process data programmatically. In Listing 1, a sample cursor shows the construction of such a cursor

that compares the staging table and the main table for differences in any column. This cursor can be used to preview the differences as well as applying the changes to the database. All records having a count of 2 are identical in both staging table and the main table. When the count is 1, then a change has been detected.

The database cursor can be a global variable in a package implementation. Other control variables can be placed in the package to prevent manipulation of the table outside the package by someone footprinting the system or by someone who has broken into the database.

The procedure to update changes uses this cursor to decide what has changed and reconcile the incoming data with the rows in the `employee_main` table. The major thrust of the code is relatively simple and is shown in Listing 2.

The setting of global variables helps determine the identity of the user running the update procedure.

- Interrogating the `SYS_CONTEXT` function provides the true user name and cuts down on spoofed identities. The user name invoking the package may not be the same as the schema that owns the package.
- The FOR ... LOOP opens up the cursor previously defined in Listing 1 and plows through it a row at a time. When the value `OLDROW` appears first, it means that the corresponding row in `Employee_Main` is deleted. Behind the scenes, a trigger named `DEL_EMPLOYEE_MAIN_TRG` takes care of the other housekeeping on each row.

**Listing 4b.** PL/SQL code to retrieve records from the history table

```

101.  --
102.  -- Next part: Put records from history into collection
103.  --
104.  BEGIN
105.      SELECT * BULK COLLECT INTO restored_emain
106.      FROM employee_main_hist
107.      WHERE updated_date = undo_time;
108.  EXCEPTION
109.      WHEN NO_DATA_FOUND THEN
110.          v_code := 0;
111.          v_errm := v_module_name||' - NO DATA FOUND to restore. Continuing ';
112.      WHEN OTHERS THEN
113.          v_code := sqlcode;
114.          v_errm := sqlerrm;
115.          g_employee_state := NULL;
116.          RAISE;
117.  END;
```

- The only other case is that a *NEWROW* is in the incoming data. The insert statement places it in the main table along with audit columns showing what user caused the insert statement and when it occurred.
- Along the way, counters are updated (iold and inew) when changes occur.

The outright deletion of an employee record may make data managers nervous, but a closely bound database trigger on the `employee_main` table carefully controls what takes place, as illustrated in Listing 3.

- First, at line 7, before any deletions take place, a copy of the record ships off to the `employee_main_hist` table. This event becomes part of the audit

trail, and all the values are stored along with a timestamp and a user name tied to the change.

- Second, additional protections are in place on lines 17-18. The deletion is not permitted unless the packaged global variable state is either *UPDATING* (the usual case) or *REVERTING*. A user attempting to sabotage the data center by trashing the `Employee_Main` table would not be able to do it by deleting the records. An error message would come up. An even better response for secure coding would be to send an urgent notification to the DBAs that an unauthorized deletion was attempted.

Similar triggers to prevent insert, update, and delete operations placed on the staging table, main table,

**Listing 4c.** PL/SQL code to reinsert previously removed records into the main table

```

118.      --
119.      -- Last part: Put records from collection back to employee main table
120.      --
121.      BEGIN
122.          FORALL rec IN 1 .. restored_emain.count
123.              INSERT INTO employee_main VALUES restored_emain(rec);
124.          COMMIT;
125.      EXCEPTION
126.          WHEN NO_DATA_FOUND THEN
127.              v_code := 0;
128.              v_errm := v_module_name||' - NO DATA FOUND to restore. Continuing ';
129.          WHEN OTHERS THEN
130.              v_code := sqlcode;
131.              v_errm := sqlerrm;
132.              g_employee_state := NULL;

134.      END;
135.      ELSE
136.          v_code := 0;
137.          v_errm := v_module_name||' - Could not revert data. If history table is not empty, contact DBA.';
138.          g_employee_state := NULL;
139.      END IF;
140.      p_sqlcode := v_code;
141.      p_sqlerrm := v_errm;
142.      EXCEPTION
143.          WHEN OTHERS THEN
144.              v_code := SQLCODE;
145.              v_errm := SQLERRM;
146.              dbms_output.put_line('Error encountered - ' || v_code || ': ' || v_errm );
147.              p_sqlcode := v_code;
148.              p_sqlerrm := v_errm;
149.              g_employee_state := NULL;
150.      END;

```

and history table will rollback and prevent operations from taking place unless the package variables are set exactly. This forces all operations to be handled through the database package alone.

### Programmatic Reversion Of Last Good State

Finally, being prepared with counter-measures when the black hats do invade is essential. It's better to develop responses ahead of time rather than doing so under pressure and sweating at three in the morning.

Incoming data sometimes needs to be backed out or rolled back even after all updates have been completed and committed. Sadly, this is where many software engineers and database administrators fail to plan; however, it is easy to recover without having to turn to time-consuming database restorations.

An undo procedure is not complicated to add, provided all the necessary ingredients are present: main table, history table, and package variables to signal and control the operations. Listings 4a, 4b, and 4c show how to revert to the previous database state.

Back to our scenario – briefly, to restore the previous state, the most recent updates to the main table must be discarded, then the last previous state must be restored from the history tables. The first part, including important definitions is shown in Listing 4a.

At lines 13 and 14, a data structure to hold a collection of the history records is initialized.

- At line 24, an important package variable is set to `REVERTING`. Working in concert with the `DEL_EMPLOYEE_MAIN_TRG` database trigger this makes sure that the deletions don't go straight to the history table.
- Line 27 calls a function to determine the last transaction date. This `undo_time` identifies the set of last updated, inserted, or modified that to purge from the main table. By design, the timestamp is the same one applied to the records changed and stored away in the history table.
- Line 56 uses the `undo_time` to identify records to restore from the history table. These records are stored in-memory into the collection called `restored_emain`.
- An exception handler, coded between lines 57-66, executes in situations where no data could be found to restore or other error conditions arise.

Finally, we restore the records in the history table back to the main table, effecting a full recovery as seen in Listing 4c.

- Lines 71-72 in Listing 4c take the in-memory collection of values from the history table and put them back into the main table.

### Author's note

These procedures assure greater data security when taken together to shield the original uncorrupted state from purposeful or unintentional data compromise. Creating this protocol ahead of the possible incursions insures better outcomes and recovery, diminishes the risk of data loss, and avoids lengthy untried manual intervention to gain command of the database. These protocols have been created as the result of a real situation at the workplace requiring great rending of breast and tearing of hair, far into the night.

- The rest of the code in Listing 4c consists of exception handling and resetting the package state for the database session in case other packaged procedures are called upon to manipulate the employee staging, main, or history tables.

### Final Thoughts On Securing Coding Processes

Code must be ready and tested to remove changes that were mistakenly applied and made to return to the previous uncorrupted state, whether the changes are the result of an incomplete data transmission, power outage, running out of space on storage devices, or an *oops!* on the part of the data supplier or data center staff.

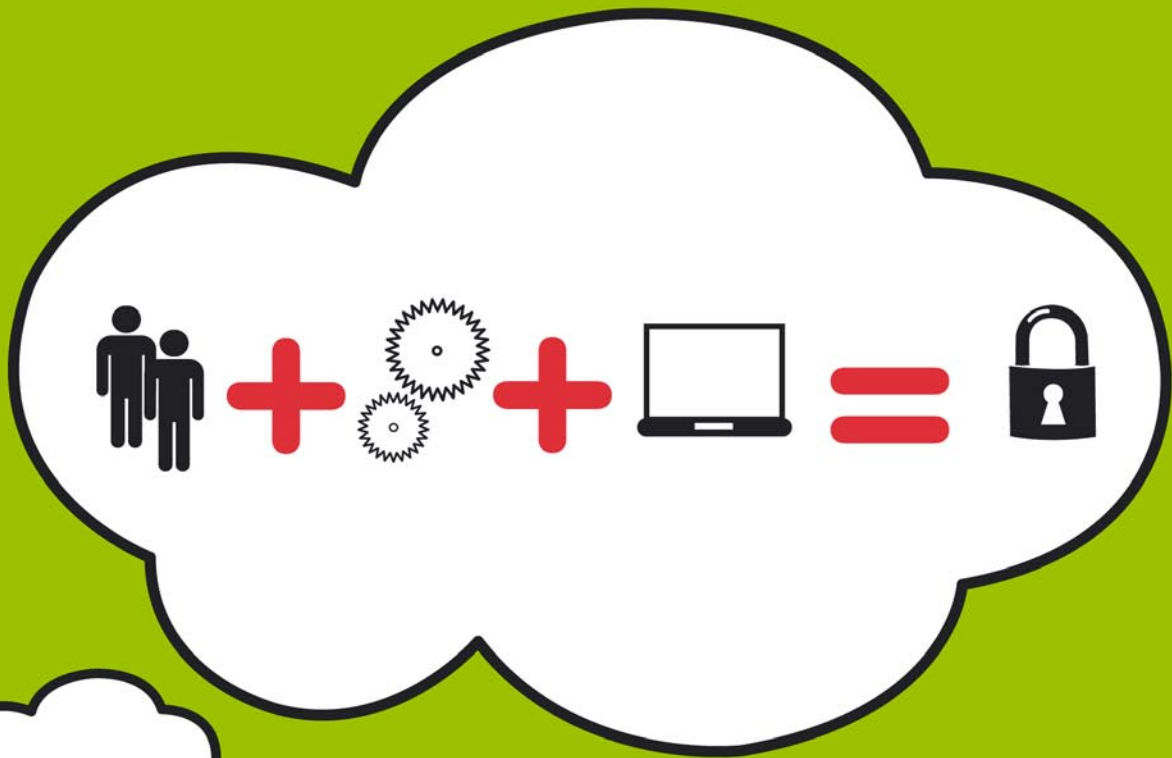
Other handy tools include the ability to preview changes before allowing them into the database. The code to preview changes can re-use the cursor code from Listing 1.

Additionally, being able to view differences is a plus to ascertain whether to edit or transform a faulty data file to eliminate errors prior to reloading. These may become necessary as it's important to rectify data errors quickly before the next scheduled data feed arrives.

---

### STEVE HODGE

*Steve Hodge is a database professional living in the United States. His background spans software engineering, data architecture, practical performance analysis and operating systems. In his spare time, he enjoys skateboarding and being around people with good senses of humor.*



# INFOSECURITY RUSSIA. STORAGE EXPO. DOCUMATION'2012 – RUSSIA'S PREMIER EVENT

The IX-th International Exhibition InfosecurityRussia. StorageExpo. Documation' 2012 ensures maximum usefulness of the visit for attendees and the highest ROI in Russia for exhibitors.

Register free to attend now at:

[www.infosecurityrussia.ru](http://www.infosecurityrussia.ru)

The premier event for the markets of information security, data storage, electronic document management and state electronic services in Russia.

**28 – 30 September 2012**

Sokolniki Expo  
& Cultural Centre, Hall 4

Moscow Russia

**infosecurity**  
RUSSIA

**STORAGE  
EXPO**

**DOCUMATION**

Organised by:

**Groteck**  
Business Media



# Mobile and Tablet Application Coding Security

In this article we will attempt to discuss briefly some of the main mobile app security issues of today and consider what developers have to do to maintain and improve their coding security practices.

**M**obile applications (apps) are being downloaded billions of times each year through a huge array of mobile stores, the most popular being; Apple's App Store, Windows Marketplace for Mobile and Android Marketplace. As with the traditional pc environment, mobile app publishers and developers have an endless job in securing their source code from malicious code. Those of us that write code know the difficulties we face just keeping up with the exploits – ask Microsoft and Apple!

Traditional security applications that ensure mobile code security include techniques such as cryptography, code instrumentation and the use of the system kernel as a reference monitor. More recent approaches combine some of the traditional ideas with new, more sophisticated ideas to enhance mobile code security. These new approaches include the techniques of sandboxing, code-signing, firewalling and *proof carrying code* (PCC).

Malicious code will look to exploit weaknesses by changing the way the code binds between coding fragments and the location of where the code is to be executed. The ability to relocate code – for example a virus attaching itself to an existing code is very evident in today's coding world.

## Secure app Coding Techniques

There are practical techniques to securing app code – the first involves limiting privileges to a set of operations – this is known as sandboxing (more on this later). The second technique involves identifying executables as they enter the trusted domain – aka firewall approach – do you want the app to run and how will it run are important queries. The third technique involves code trust – is the executable trustworthy?

– This is what we called code signing. Lastly, the code needs proof that properties have been met. All are equally important techniques when looking to secure the app code.

Unfortunately, as with most apps, performance is sacrificed for increased security. This field is attracting research as I write, but in the current economic climate, investment into this potentially highly profitable USP remains elusive. One can only hope that this will improve over time.

So what about cryptography? Most readers will know what cryptography is, so I will not explain here. What we can say is that cryptographic methods can be used to encipher data exchanges and identify and authenticate users, agents and platforms.

Now let's take a look at some other coding protection techniques i.e. kernel; coding instrumentation etc. More will be discussed on cryptography in the code signing below. The kernel acts as a proxy between processes and system critical operations which means the kernel can monitor all access, enforce safety security policies and prevent un-trusted code from corrupting the system (this is what the NovaShield project has been developing aka – anti-malware behavior detection). There is performance degradation here though, mostly to do with the overhead of passing parameters.

So what about coding instrumentation? This basically means modifying un-trusted code so that it can be monitored at runtime. Code that runs normally and that does not violate a security policy will run normally with no changes to its behavior. If a violation does occur then the system will be notified and the app will be terminated or contained, so that it doesn't affect another part of the system i.e. sandboxing. The major advantage of code instrumentation is that it can be executed in isolation

and no extra information about the code is required. The major issue with this is the performance – there is a substantial runtime overhead because a runtime check is required for every critical operation. This trust model places the runtime monitoring inside the trusted code base, while the mobile code is placed outside of the trust zone.

### Sandboxing Code and app Privileges

Sandboxing is used in software to quarantine module faults, in effect placing them into isolation. This allows un-trusted apps written in say an unsafe language to be executed safely within a single virtual address space of app. Un-trusted systems interpretable code modules are transformed, so that all the memory accesses are confined to code and data segments with their system fault domain. Access to the system resources can then be controlled through a unique identifier associated with each domain – this is what is called *sandboxing*.

Sandboxing will allow foreign code to execute within the sandbox in the host operating system. The code can be controlled by allowing monitored access to local host resources i.e. RAM, CPU etc which by its very nature will stop malicious code from delivering its payload (overflowing system resources i.e. DoS attack). In my opinion this technique should be considered when building mobile apps.

### App Code Signing

Nokia were the first (through Symbian) and now Windows Mobile 7 and popular app stores such as Windows Marketplace for Mobile have implemented code signing technology to address the obvious security concerns. The idea is that a platform (i.e. Windows) use code signing to control an app that is allowed on a network, taking specific measures to ensure the safety of a mobile app for users and the networks upon which they rely.

Code signing is done by the developer of the code with an example being a developer obtains assurance that the source code is trusted and that the code hasn't been tampered with (think Symbian or Windows Marketplace). The code signing affectively verifies authenticity and integrity (security layer protocols). This model is based on *trust*, so as what happened with Symbian, some rogue apps did get through the code signing process. However it's worth noting that any rogue apps were pulled from the Nokia store very quickly.

Windows Marketplace is now requiring code signing technology (Google – anyone listening from Android?) that essentially signs the mobile app code with a digital signature, creating a digital wrapper that both validates the source code and confirms that the source

code has not been modified. Code signing by its very nature uses a public cryptographic key. A developer or software house will use the private key to add a digital signature to a piece of software code. Windows 7 for example will use the public key to validate the signature during the app download process and compare the hash used to sign the app against the hash of the download app. This is a very simple code signing process.

Mobile code signing should actually feature two digital certificates .One for identifying the publisher and the other to identify the app content. The publisher could use the publisher ID to sign the code and then uploads to the network for validation to a *Certificate Authority (CA)* for code signing. The signature is then validated and a unique content ID would be generated with the validated publisher ID and the code can be then be *trusted* or *not trusted* for app store inclusion.

Another headache is the API's, especially given Windows Phone 7 uses Windows Privileged access for Marketplace – so in this instance a third-party validation would be required to validate the content ID. The major problem right now is that some app store providers allow only signed apps while others require code signing in order for apps to have access to a sensitive API. In this instance, if an app doesn't recognize an apps digital signature as being valid, then the app will not run. This re-signing process is something that all app stores should consider – Apple does something similar with their App Store.

### Windows Mobile – Code Obfuscation

Microsoft recommends the use of a technique called code obfuscation, which uses a variety of techniques to make it harder for a hacker to decipher and recover the underlying source code. Opinions on its usefulness vary widely and sometimes wildly. In general, many programmers who use obfuscation see it as just one of the steps they can and should take to protect their applications, data, and intellectual property where protection is needed.

Obfuscation does indeed help, but it really does not present an obstacle for hackers. Most security experts (including the author of this article) agree that the most reliable way to make sure that your app doesn't leak important information is not to have that important information in your app in the first place. Should a mobile device store, codes, digital keys etc – some security experts believe this should reside in the cloud, not on the mobile device. Any app that has sensitive data locally stored should be encrypted. It's that simple.

It's not so much a case of what defensive wall you have but making sure that the data that a hacker wants

is limited or not useful. So what about the unavailable data? Hackers will always attempt to find a way to extract *unavailable* data, so this route isn't an option. Developers need to consider this when coding their apps.

### Android – Open Source Equals Open Season

Google's open policies on app approval and the availability of third-party app markets have left lots of gaping holes which malicious apps can install on a user's device. Google's Android is open source which allows for developers to create apps and share them with us. Android is developed on a Linux kernel, so developers do get to see the deep internals of the Android operating system. There seems to be a lack of centralized documentation on Android too.

Start searching the Web and you will not find one centric location where you can find the source of an error or crash. There seems to be endless forums and posts (some very out of date) containing articles that don't provide the answers. Has anyone tried to find documentation on Android on how to adapt it to a particular device? Exactly, it's very hard to find. How does a developer submit an issue or ask a question when there is no central location – the license agreement clearly states that you use Android at your own risk (hence open source).

Developers look to hack the Android system (mainly because it consumer focused unlike BlackBerry which is security focused) to resolve their coding issues, rather than follow any formal coding process. This leaves Android open to performance and security related issues. There is some evidence that some Android apps have led to Android devices to reboot several times a day, either because they crashed the OS or didn't close properly.

Then there is the issue of fake Android markets appearing in China and other countries promoting unique, pirated and redistributed apps. There are fake Apple App Stores but you can only use these stores if you have a jailbroken iPhone for example. Most users will not have jailbroken their handsets. Some security experts believe that Android should consider blocking access to fake Android markets and only allow access to the official Android market.

#### Did you know?

iPhone users don't know what data is accessed (except location) – RSA, 2011

### QNX tablet (RIM) Following in iOS Footsteps

RIM continues to be very strong in mobile secure computing. The PlayBook tablet launched earlier this year runs on the QNX *Real Time operating system* (RTOS). With a little research you can identify that

this operating system has yielded only about 75 vulnerabilities. However, most of these are over five years old. The vulnerabilities were linked to QNX Neutrino 6.5 which is the old codebase – RIM developed 6.6 and with no documentation available on this (let alone potential vulnerabilities) it's hard to speculate what the codebase changes might have been.

A recent report (August 2011) from NGS Secure stated *RIM has built a robust system on top of the existing QNX microkernel. They have restricted file and user permissions at the operating system level, leaving a reduced attack surface. The fact that some of their other technologies (such as PPS) are implemented as an abstraction on top of the file system certainly contributes to ensuring that the attack surface is minimized and that the general controls implemented to protect the file system are also effective to protect these other layers.* The BlackBerry PlayBook for example, is the first tablet to earn FIPS 140-2 certification in the US.

Playbook apps are not allowed to communicate with each other. Apps only have direct access to the direct file system which resides in a sandboxed folder. If an app needs device-related functions, it will go through the BlackBerry API for the PlayBook (but API restrictions will still be applied). Apps which are loaded via a developer mode are owned and run by the user. Each installed app is assigned their own user and group when installed. This forces each app to operate in a sandboxing mode. This coding principle (sandboxing mode) isn't unique to RIM as Apple with iOS for example, employs similar user permission and API access models. Apple's App store uses DRM and all apps that are present in the App store are encrypted.

### Patch Process – Android Case Study

Android has been making the news recently, mainly due to its open source nature. This makes Android a very interesting case study when it comes to Android vulnerabilities. Lookout Security conducted some interesting research which is worth referencing here. Lookout looked at the time it takes for a device to reach its vulnerability half-life (the time it takes from a public announcement of the vulnerability to having 50% of devices in market patched) varies significantly.

While the Exploit exploit took 42 weeks to reach its half life, CVE-2010-1807 (WebKit NaN) was patched on 50% of devices in 30 weeks. There are a handful of vulnerabilities that have yet to reach their half-life, for example [RageAgainstTheCage](#) has gone 40 weeks and counting, and 55% of devices remain vulnerable as of July 2011.

Factors that affect the length of patch cycles include:

- Time it takes Google to release the patch to the Android Open Source Project repository.
- The level of commitment by OEM manufacturers and carriers to update devices with the latest release.
- The number of customizations on devices and time it takes to flash each firmware build with the updated OS release.

While great strides have been made in patching vulnerable devices, we see opportunities to continue to improve the patching cycle.

- Manufacturers should take advantage of the Compatibility Test Suites to cherry pick fixes for their builds.
- The industry should continue to build tools to track and manage patch levels (particularly in the enterprise).
- Carriers and OEMs should make security a first class-feature in releases, ensuring that the latest patches are always included.
- Players in the ecosystem should agree to unlock bootloaders, thus eliminating the conflict of interest between vulnerability disclosure and the ability for users to control their own devices.

While the patch cycles for mobile operating systems are typically longer than that of the PC, there are signs that it is getting better. Android CTS has proven to help drive adoption of critical updates. For example, several prominent devices shipped in the last few months have been shipped with critical security fixes. Until everyone in the mobile ecosystem considers security a top priority by accelerating patch cycles and removing the conflict of interest for vulnerability disclosure, longer patch cycles, and thus vulnerable devices, will continue to persist. We're hopeful that mobile device patching will continue to improve and eventually even improve on the PC status quo.

*Reference: Lookout Mobile Security, August 4th, 2011 blog post*

### **JavaScript and ActiveX Mobile Security**

Readers will know that there is an ever increasing number of mobile websites that are using/and considering using JavaScript and ActiveX. Incorrectly designed or poorly written ActiveX controls can cause serious security problems (XSS attacks) in two container types, Web (mobile) browsers and e-mail clients, because Web pages can invoke ActiveX controls by using HTML or a scripting language and e-mail applications can often display HTML-formatted

# Join

## hakin9 team!



If you would like to help our team in creating hakin9 magazine you can join our authors or betatesters today!

All you need to do, is to send an email to:

**[editors@hakin9.org](mailto:editors@hakin9.org)**

and give us a brief description of your field of interest.

**We look forward to hearing from you!**



text, which means that e-mail messages can also invoke ActiveX controls.

ActiveX controls how web pages display animation, audio and video – the ActiveX control is stored in the browser cache or on the hard drive. Other apps can have access to this Active X control. The Active documents allow users to view for example Microsoft Office 365 (cloud-based service) documents within a web browser. ActiveX scripting resides in the browser so that the browser (IE for example) can run Java Applets.

Using JavaScript and ActiveX allows websites to provide additional functionality that cannot be handled by HTML (although HTML 5 does handle this). JavaScript, ActiveX, Macromedia Flash and Shockwave can be accessed across a private or public network and executed remotely – this is what we call *mobile code*. Unfortunately, it is entirely possible to (mis)use Java, especially in its applet form, as a vehicle for attacking systems. Language-based security controls like those found in JavaScript make writing a hostile applet more difficult than it might be otherwise, but they don't make it impossible.

### Did you know?

If you mark a control for scripting, you might want to allow the control to be scripted only when invoked from a specific restricted domain.

There are two approaches to securing JavaScript – these are called Sandboxing and Code Signing. The sandbox model (described earlier) restricts access to a limited amount of resources or files on a users system. The code signing model is based upon a third-party company who will sign the code with a digital signature. One of the most popular uses of code signing is the use of secure SSL (secure socket layer) – you will see this on websites and with some web based apps. Within an SSL session, the network connection will ensure the code has not been tampered with. Users can view (most don't know where to look or what to look for) the certificate (by clicking the padlock icon in the browser) which will display the details of the code signing.

The main issues developers have found with the Java sandbox is that it is a little too restrictive, though some developers do decide on developing their app code outside of the Java sandbox. Java applets by their very nature can scan a user's file system, modify files, access memory and open other apps.

Readers will know that malware writers have created hostile Java applets which will modify system and app files – this in itself continues to pose a security threat. In the final section we will look at the emergence of HTML5 and whether it is as secure as some say it is.

Further reading: Secure coding guidelines for Java programming: <http://www.oracle.com/technetwork/java/javase/tech/seccodeguide-139067.html>.

### HTML5 Mobile app Security – the ENISA View of HTML5

This month (August), the *European Union's computer security agency* (ENISA) warned that HTML5 coding implementation standards under development as part of HTML5 were undergoing a rewrite that may omit important security issues. ENISA looked at thirteen specifications (in a 61 page document) within HTML5 and found 51 *security issues*. Some of the issues appear that they can be fixed by tweaking the specifications, while others are more risk based on the features that users should be alerted to. One particular part of the HTML5 specification involves the *submit* button which allows for a web-based form to be placed anywhere on a Web page. This means an attacker could inject other HTML onto the page, such as a different form button, and then cause the information in the form to be sent to the attacker rather than the legitimate website. HTML5 mobile app developers should read the ENISA document above.

### Final Thoughts on the Development Cycle

Code signing and sandboxing are two app security principles that should be proactively incorporated into the mobile coding development cycle. It provides users and network providers with significant confidence that apps are safe to download, install and run. Empowering developers to protect end users should be first and foremost as this has significant links to brand reputation and market value. Protecting and securing your app code both at the network (this also includes within the browser) and client level has to be the priority.

---

### JULIAN EVANS

*Julian Evans is an internet security entrepreneur and Managing Director of education and awareness company ID Theft Protect. IDTP leads the way in providing identity protection solutions to consumers and also works with large corporate companies on business strategy within the sector on a worldwide basis. Julian is a leading global information security and identity fraud expert who is referenced by many leading industry publications.*

The background of the entire page is a dark blue/black color with several bright white lightning bolts striking downwards. The bolts are jagged and vary in thickness, creating a dramatic, high-contrast effect.

In the next issue of  
**HAKIN9** magazine:

## **Hacking Apple**

Available to download  
on **September 30<sup>th</sup>**

Soon in Hakin9!

TOR Project, Botnets, Social Network Security, Hacking Apple, Biometrics, Rootkits, Debugging/  
Fuzzing, SQL Injection, Stuxnet, Hacking Facebook, Port scanner, IP scanners, ISMS, Security  
Policy, Data Recovery, Data Protection Act, Single Sign On, Standards and Certificates, Biometrics,  
E-discovery, Identity Management, SSL Certificate, Data Loss Prevention, Sharepoint Security,  
Wordpress Security

If you would like to contact Hakin9 team, just send an email to  
[en@hakin9.org](mailto:en@hakin9.org). We will reply a.s.a.p.

# VirusTotal

Hispacec Sistemas has managed the service, VirusTotal, since 1<sup>st</sup> June 2004. The website (<http://www.virustotal.com>) offers the public access to multiple Antivirus (AV) engines hosted by them to provision online scanning of individual files to uncover malware by harnessing a combination of signature-based and heuristic detection.

**V**irusTotal has since evolved from its humble beginnings to also offer analysis of website content with malicious intent (ie. *Phishing, Spam, malware*).

Their online Analysis tool is available to any *Operating System* (OS) with a browser and *Javascript* enabled.

Uploading an *Eicar* test file to the website triggers warnings as expected. In situations where a specific file uploaded has been tested before, it may be advisable to click *Reanalyse* to rescan it.

*Hot Toys* is a toy manufacturer from Hong Kong that has had its website (<http://www.hottoys.com.hk>) compromised previously. Its website is used as a test case of the web analysis engines hosted with *VirusTotal*. Two engines flagged the website as malicious whilst one was inconclusive. This warrants caution to avoid visiting the website and having one's system infected.

Harnessing the search power of *Google*, the search engine also marks the website as risky to confirm the warnings from the *VirusTotal* scan.

*VTzilla* is a *Firefox* plugin that can be engaged to scan URLs. It can be downloaded (<http://virustotal.hispasecsistemas.netdna-cdn.com/progs/vtzilla.xpi>) and installed to any OS with *Firefox* and *Java* available. The caveat of this plugin is that other tools can affect its effectiveness in detecting disputable websites. Another *Firefox* plugin, *HTTPS-Everywhere*, encrypts all my *Google* search results and thus influences the scanning of the *Hot Toys* URL displayed in my *Google* search.

*Phishing* sites do not escape the examination of *VirusTotal*.

*Windows* users have the option of installing the *VirusTotal Uploader* tool. This software allows them to select a file residing on their system directly to the

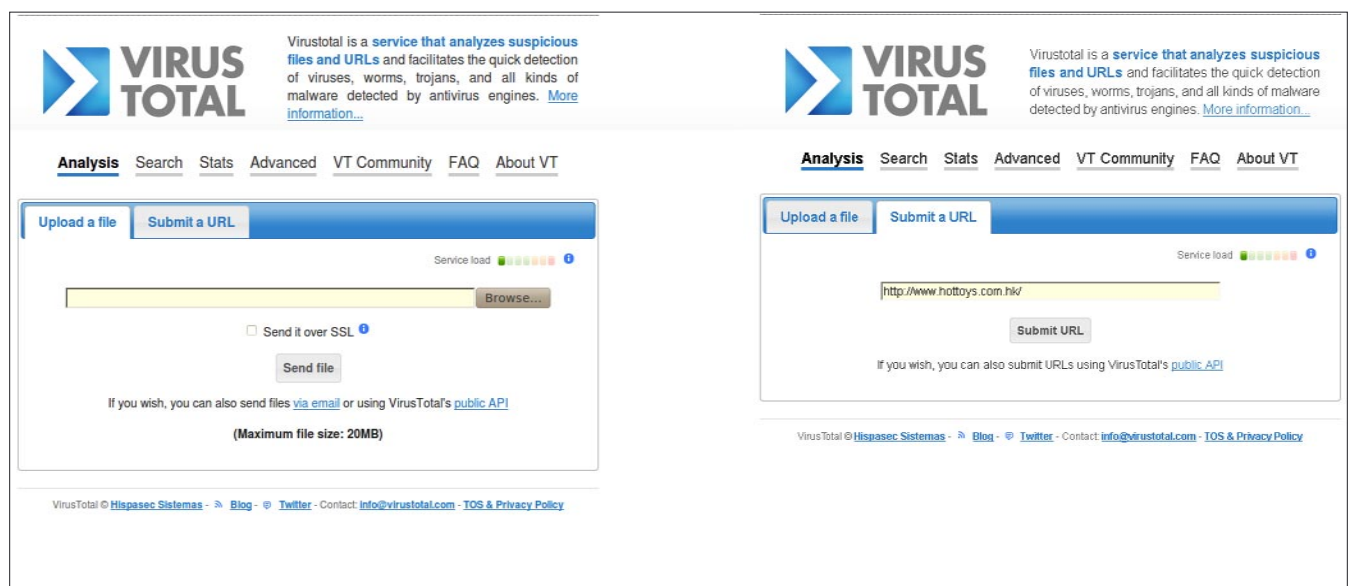


Figure 1. Analysis section





# What's Wrong With The Bible?

Corporate IT security policies are often described by security professionals as “the Bible”. This comparison always makes my skin crawl, since it suggests a certain lack of imagination. But in reality, the comparison makes sense. Both interpretations were probably written a long time ago by people who hadn't met you, or by employees that faced precisely the same issues, technologies, and situations you face in your job today. More than that, both were probably written by different groups of people over time.

**T**he Bible doesn't talk about social media – there is no mention of Facebook in the book of Leviticus while there is a good chance an organisation's security policy doesn't address social media precisely either.

In a 2010 poll, conducted by a security consultancy, including twenty one organisations, asked one simple question: does your organisation have a social media policy? The results were illuminating: more than half of the organisations did not have such policy in place, and only one third had their organisation's policy crystallized into a single document, and 14% of respondents had references to social media scattered through a range of sources.

In particular, very few people have really read, understood and retained the entirety of either text and in the absence of a commonly agreed, authoritative understanding, an array of interpretations are possible, many of them definitely not what the authors intended, and quite possibly the cause of what is, objectively, wrongdoing.

None of this, however, should be interpreted that security policy is not relevant. It is the one repository that lays out an important framework for the organisation's security posture. It is vital for demonstrating compliance with regulation and legislation. For example the set of requirements under PCI-DSS (which affects all organisations which handle or process card data). It is also important for handling difficult questions from external auditors. Most importantly, it is the one document that tells you, the organisation's employee, customer, or partner, what is expected of you, and what

can be expected of them. So why are security policies so often badly written, to the point of being unusable?

So then, how can such a critical document go so badly wrong in so many organisations? There are a number of things to consider. Many organisations, particularly in Western Europe and Japan, base their security policies on *best practice* industry standards, such as ISO 27000. These are great standards, in that they provide an internationally recognised baseline. However, few people would argue that the International Standards Organisation produces documents that are fun and easy to read. Additionally, the roots of the ISO 27000 series lie in the old British Standard, BS7799. Although published in 1995, this standard was derived from 1980s trends in the UK government. In other words, the *best practice* standards were in essence written twenty five years ago: the meaning of *best practices* was very different then.

Another consideration is that security policies are often written by multiple stakeholders. There is the old saying that a camel is a horse designed by a committee. Perhaps more aptly, we should consider the old parlor game *Consequences*, where a story is written line by line by a succession of people unable to see what was written before or after them. An acceptable security policy will cover a lot of ground; after all, it is meant to be the definitive documentation of the organisation's security stance.

So, in any given organisation, which department will not want to have some input? After all, they will all be affected. The Human Resources department feels it needs to have

some input about personnel and disciplinary issues. The Legal department will need to make sure the wording is appropriate. And of course, the IT department will have a lot to say; there will always be a million reasons why *best practices* cannot sensibly apply to each one of them. Perhaps a particular issue has caught the attention of the organisation's executives and would eventually be reflected in a given policy. At some point in time, the organisation will encounter a situation where complying with its own policy, would be technically unfeasible, too prohibitively costly or inefficient. So having gone through all this elaborate exercise, to produce a policy that is based on *best practice* (even if it is out-dated) including an exceptional amount of expert input, what is the organisation left with? The balance is something that doesn't really mean very much to precisely the people who need it most: the ordinary employees. Most people, most of the time, want to do the right thing. However, what happens if they don't know what the right thing is? After all, information security is a fairly complex and broad field – but the policy has been written by a committee of technical experts, not with an ordinary person in mind. After all this effort, what should be a highly valuable resource for everyone to use rests, unused and ignored on the metaphorical shelf – and security problems occur again and again, because someone, somewhere in the organisation is not doing what was committed to in the policy. So how can organisations break free from this dismal situation? Three simple things would make a huge difference:

- Real thought about how the purpose and scope is presented
- Keeping out what is not policy
- Keeping current

### It's A Policy, But Not As We Know It

Test the policy. Can ordinary people understand it? If they don't, that is a big problem. An organisation needs to make sure its policy is fit-for-purpose; if the readership can't understand or interpret it, it's fallen at the first hurdle. Is it a language problem? Or is it the format? This is a tricky problem; if a security policy is written in the traditional format, like a technical document, it is not searchable and hard to read by a business-minded audience. So who really wants to wade through fifty or a hundred pages of something that is irrelevant to their needs to find the two sentences that matter to them for the particular question they have? Death to the policy document, I say. What people need is a searchable database.

### Brutal Simplicity

Some things just don't belong in an organisation's information security policy, however most of the IT groups want it to be there. The policy should be technology agnostic, so as technology changes, the policy stays sound

and provide a baseline for any situation. In other words, it should communicate to employees what the company's baseline is. This tends to point towards a simpler document, rather than the three hundred page magnum opus typical of, say, a large European or American bank. Anything which is conditional (*what if..?* type questions), technical (which ports are allowed, technical builds, etc.), or transitory (won't apply in the foreseeable future) doesn't belong in a policy document. A lower level document, perhaps a technical standard or procedure is the most appropriate document to include such details; but not in the policy because it clouds and muddles the purpose of the document.

### Staying Hip With The Kids

Many things are often not specifically considered in a security policy. Ignoring a social media policy rather suggests some confused thinking. An organisation may block access from the desktop, but what about the phones and mobile devices that your staff carry? And if the organisation's leadership have a desire to limit staff from posting negative comments online about the organization, how will you control what people do when they get home? In the absence of a clearly articulated policy, how will your staff know what is expected of them?

GreyReview estimates that, taking into consideration unique Twitter accounts and all the users of all the various Twitter APIs, the total number of users in the Twitter ecosystem is approximately 240 million. It is estimated that the UK alone has upwards of 27 million Facebook users – a number that continues to grow. You might not like it or use it, but what about your staff? What about your customers? Deliberate or inadvertent sharing of information which organisations would rather keep confidential, is considerably easier by using any kind of social media. To this effect, certain branches of the US military, for example the US Marine Corps, have developed and published very specific guidelines for *unofficial* social media postings.

### How To Cope With Change

To paraphrase Douglas Adams, what was invented before we were fifteen, we tend to think of as simply the state of nature. What is invented between when we are fifteen and in mid thirties, we think of it as new and exciting. And what is invented after our thirties we tend to think of it as dangerous and bad, something that should be stopped. How could anyone write a rule book to cope with all situations across all three stages? Lots of things are bound to change. But a simple, easily understandable guide is a possibility, just think about the Ten Commandments.

---

### DRAKE

*Drake has worked on information security and strategy with government agencies, the military, financial institutions and other blue chip organisations in Europe, the Middle East, and Africa since Boris Yeltsin was President.*

# Passware Password Recovery Kit Forensic 11.0

Passware Password Recovery Kit Forensic 11.0 is a handy all-in-one package for recovering different types of passwords quickly and with ease. Be it from a Windows laptop, Mac VM, or USB stick this software raises the bar for password cracking.

## Homepage

<http://www.lostpassword.com>

## Review Version

Passware Password Recovery Kit Forensic 11.0 Build 3631 2011/08/08 (Figure 1).

## Installation

The MSI is a simple standard 8-screen installer (Figure 2) with standard EULA acceptance, Destination Folder and Product Key fields. During installation it verifies that it will need 219MB to install. Once the install is complete the user is prompted whether to start the application right away and also to subscribe to the free Password Recovery newsletter.

## Post Installation

After the install has completed the user can click on the Passware Kit icon (Figure 3) to start the application. Also the Start Menu is populated with a Passware folder which contains the Passware Kit 11.0 sub-folder containing the main application, help, and the Search Index Examiner 3.0 (out of scope for this review).

## General Settings and Configuration

The file menu contains 4 sub-menus, File, View, Tools and Help.

*File* contains the gamut of options available via buttons on the main pane as well as the shortcut keys for these functions. The unique entry is Create Portable Version to create a portable version of the installed application



Figure 1. About The Program



Figure 2. Installer



Figure 3. Passware Kit Icon



Figure 4. Options

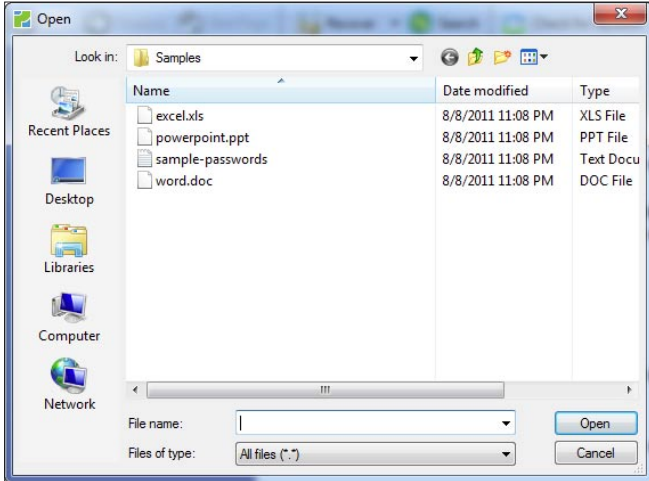


Figure 5. Samples

(on a USB key). View contains one selectable option; the Status Bar. Tools contains additional options. Options (Figure 4) for enabling GPU acceleration, Distributed Password Recovery. The product License Manager. (This is where the user can add additional agents for distributed and cloud computing).

Help contains standard information such as contextual, searchable help and index information. Links to the lostpassword website for additional support, updates and version information.

## Features

Due to time constraints some features that this product offers are not covered in this review. They are as follows:

- Recover Internet and Network Passwords. *Recover passwords for websites, network connections, and email accounts.*
- Reset Windows Administrator Password. *Create a bootable CD-ROM that instantly resets Windows Admin password.*
- Search for Protected Files. *Scan computer for protected files. Additionally Distributed Password Recovery and Bitlocker portions are not covered.*

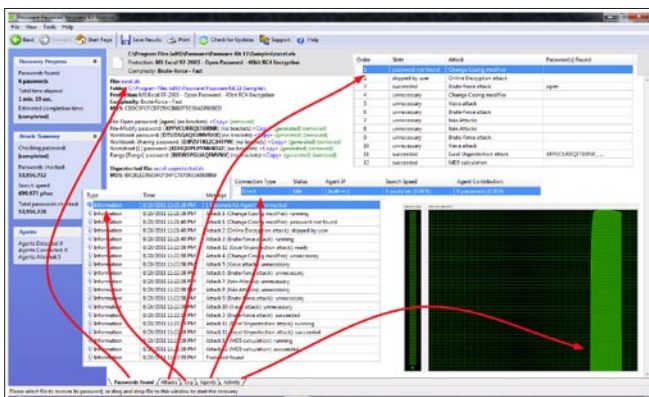


Figure 6. Recover File Password

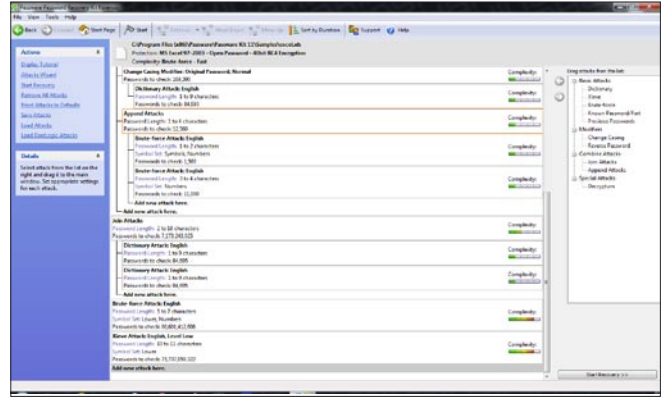


Figure 7. Recover File Password Configuration

Features that are covered in this review are described below.

## Recover File Password

*Pick a protected file to start password recovery.*

Initiating this prompts the user to browse to a protected file (single file, no batches). A few samples are available in the default directory (Figure 5) in the example pictured here (Figure 6) the mentioned tabbed information is available (extrapolated for readability). This application supports over 200 file types of encrypted files including vanilla zip and pgg zip archive. Naturally based on what type of password/passphrase was used in conjunction with hardware and distributed agents it could take a matter of seconds to days.

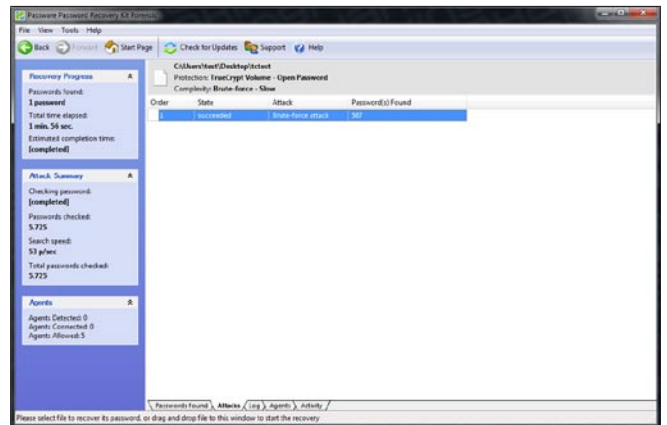


Figure 8. Unmounted Brute Force Attack

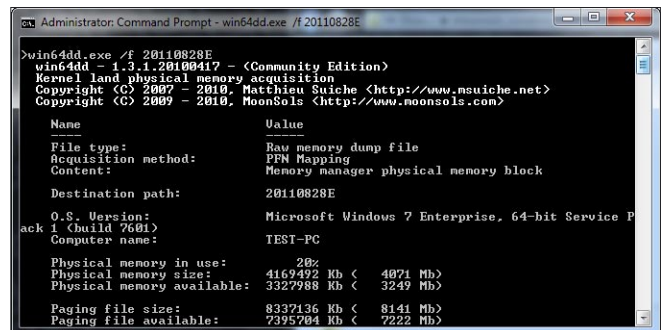


Figure 9. Mounted Attack (win64dd)



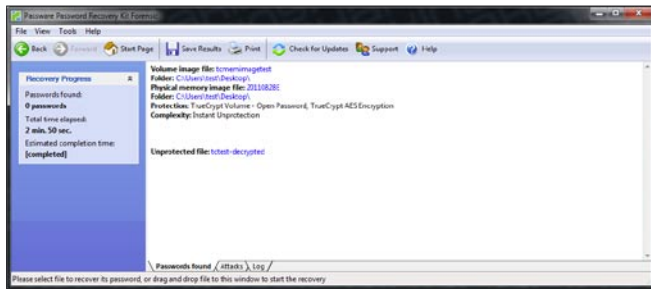


Figure 10. TrueCrypt Volume Decrypted

Going through the configuration wizard allows the user to refine the target options such as estimated length, language, alphanumerics, symbols, patterns. Something quite familiar in attack tools that could have nearly infinite parameters. The more you know about the target files the better chance have at cracking it (Figure 7).

The tests performed here were all under 10 minutes and aside from the samples most passwords were as simple as 123. In all cases the passwords were recovered successfully.

## Recover Hard Disk Password

Recover encryption keys or passwords to unlock BitLocker and TrueCrypt drives.

TrueCrypt Volume Password Recovery was performed on a few test volumes. Both mounted and unmounted. The Unmounted brute force attack (Figure 8) took only a matter of minutes for a 3 character password. The mounted attack was a bit trickier and usage of a memory imaging tool for windows win64dd (Figure 9)



Figure 11. Passware FireWire Memory Imager

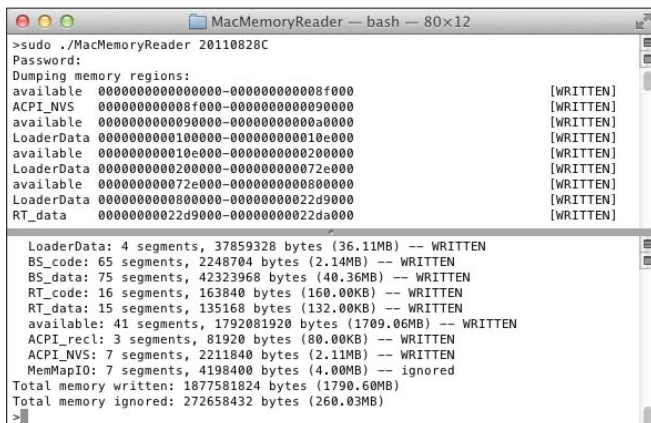


Figure 12. Mac Memory Reader

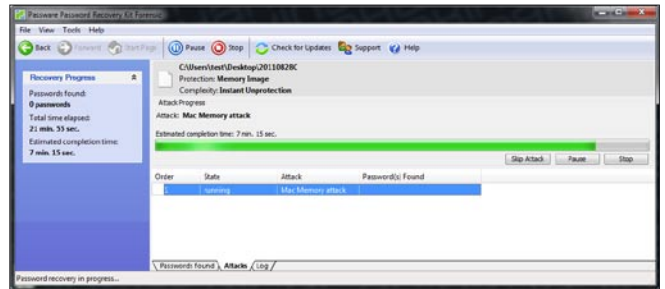


Figure 13. Mac Memory Attack Running

from the MoonSols Windows Memory Toolkit Community Edition was used to create a 5GB image from the system and then load it into the application. In under 3 minutes the 5MB TrueCrypt volume was decrypted (Figure 10)

## Recover Mac Password

Recover passwords for Mac users from a memory image.

### Note

During this review issues were encountered with Passware FireWire Memory Imager (Figure 11) on the Macbook Pro and alternate means were used to capture the image with Mac Memory Reader instead. The Macbook had bootcamp 4 loaded with Windows 7 and since it has a Firewire 800 port as does the Mac Mini that it was being tested against naturally it was believed that this could work. Unfortunately Mac, Lion, Bootcamp can't boot up using the USB key created (PFMI). At first it was thought it was the USB key model but several were tried (Kingston, Verbatim; all 4GB/8GB), then it was thought that formatting of the

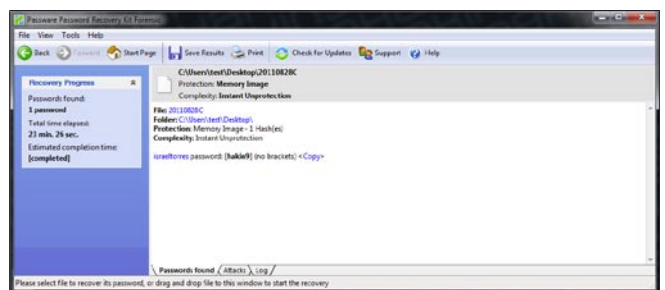


Figure 14. Mac Memory Successful

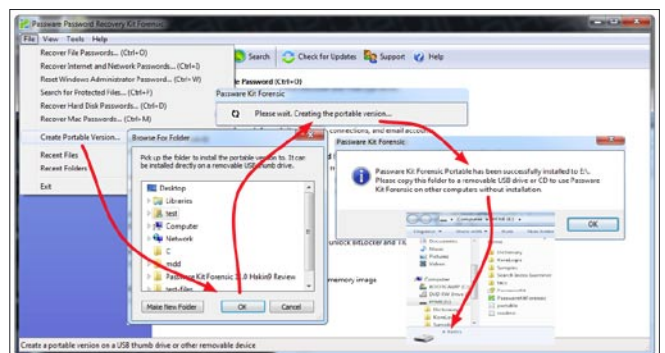


Figure 15. Portable

key had something to do with it. It appears that this type of configuration doesn't work. Trying from an older Dell also gave erroneous errors about the kernel support. It would be great to see a workaround for this using Mac versus Mac; instead of what seems PC versus Mac.

MacMemoryReader (Figure 12) was used to capture the memory image of the target Mac. Once the ~2GB image was captured it was transported via USB to the Win7 system running Passware Password Recovery Kit Forensic 11.0. Browsing to the image on the desktop the Mac Memory Attack began quickly (Figure 13) ... and resulted in the correct password (Figure 14).

## Portable Version

*USB version created from installed version.*

Creating the portable version is a simple 2 step process. In the File menu select Create Portable Version. Browser for a folder or select a USB key. The whole idea is to have this on something portable and the USB key suits fine. If you have enough space you can copy it onto your PFMI created USB key, but to be more portable it is best to keep those two features on separate keys (Figure 15).

## Uninstallation

Passware Kit Forensic 11.0 it requires to be uninstalled from the Windows Control Panel (*Control Panel->Programs->Uninstall a program*). The uninstall process runs pretty smooth and only leaves a few residual artifacts in the Program Files -> Passware subdirectory (more residual if you saved your work in additional subdirectories within).

## GUI

The Graphical User Interface for Passware Kit Forensic 11.0 (Figure 16) is quite intuitively modern to a Windows 7 environment. The sub-applications, features and functions are available redundantly allowing users to adapt accordingly. Passware offers great documentation and a traditional Windows help system. Most of the sub-applications contained within the single pane of glass

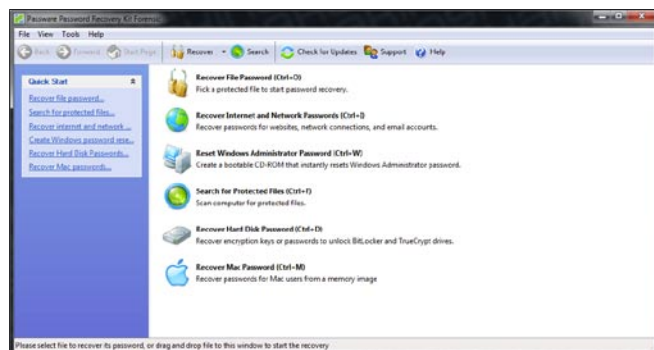


Figure 16. Graphical User Interface for Passware Kit Forensic 11.0

## Web Links and References

<http://www.lostpassword.com/kit-forensic.htm>

<http://www.lostpassword.com/support/faq.htm>

## Passware FireWire Memory Imager

[http://www.youtube.com/watch?v=l2hYlkc6\\_Pg](http://www.youtube.com/watch?v=l2hYlkc6_Pg)

...additional tools used mentioned in the review:

Mac Memory Reader (2.0.4 [released July 2011])

[http://download.atc-nycorp.com/utilities/MacMemoryReader\\_2.0.4.tar.gz](http://download.atc-nycorp.com/utilities/MacMemoryReader_2.0.4.tar.gz)

Homepage: <http://www.cybermarshal.com/index.php/cybermarshal-utilities/mac-memory-reader>

Win32dd -> MoonSols Windows Memory Toolkit Community Edition

(win64dd.exe 1.3.1.20100417)

Homepage: <http://www.moonsols.com/windows-memory-toolkit/>

use sheet style tabs (i.e. Passwords found, Attacks, Log, Agents, Activity) and the overall functionality makes easy to move around. Resultant data offers a copy to clipboard function and URI data is a click away. Any time the scope is changed and where data could be lost the user is presented whether they want to clear results and return to the start page (which is the default page when the application opens).

## Support FAQ

Passware Kit Forensic 11.0 comes with a 1 Year subscription and \$395 to renew (where updates and upgrades are available). On their support page they have an FAQ, videos. For further involved technical support the user needs to fill and submit a web form to be added to the ticket queue. The knowledge base itself has three items listed as of this review.

## Price

Passware Kit Forensic \$995 with 1 Year Subscription Free and \$395 to renew: <http://www.lostpassword.com/support/updates-and-upgrades.htm>.

## Rating

4/5 – Really aside from not being able to use the Passware FireWire Memory Imager on a Macbook Pro to attack another Macbook via Firewire the product itself has a great feature set and is quite functional both for the office and the field.

## ISRAEL TORRES

*Israel Torres is a hacker at large with interests in the hacking realm.*

[hakin9@israeltorres.org](mailto:hakin9@israeltorres.org) [http://twitter.com/israel\\_torres](http://twitter.com/israel_torres)

<https://plus.google.com/102921309581624765133/posts>

*Got More Time Than Money?*

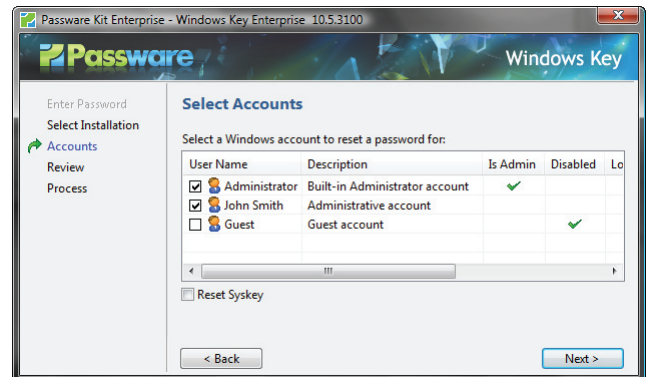
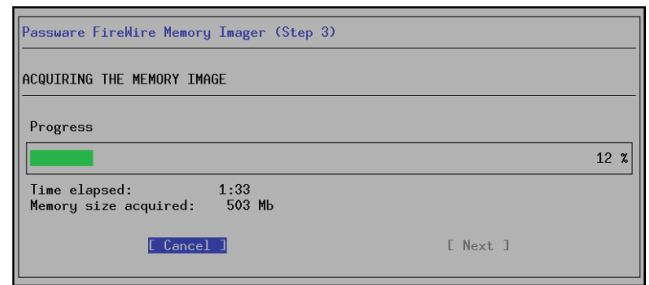
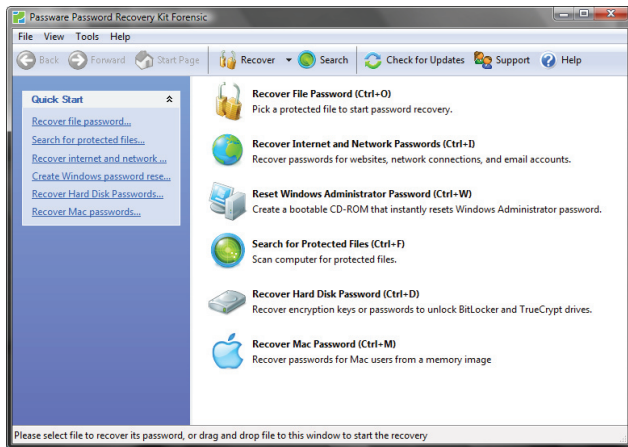
*Try this month's crypto challenge: <http://hakin9.israeltorres.org>*

# Passware Password Recovery Kit Forensic 11.0

*A Complete Password Recovery and E-Discovery Solution for Computer Forensics*

Now with Mac User Password Recovery!

Passware Kit Forensic includes over 30 password recovery tools, Encryption Analyzer Professional, Search Index Examiner, FireWire Memory Imager, and a Portable Version to provide immediate password recovery for any protected file detected on a PC or over the network while scanning. It recovers or resets passwords for more than 200 different types of files, as well as decrypts hard drives, PGP archives, and unlocks Windows 7 and Mac OS Lion Administrator accounts. Many types of passwords are recovered or reset instantly.



## Key Features

- Scans computers and network for password-protected files
- Recovers passwords for **200+ file types** Updated
- Unlocks hard drives protected with **BitLocker** and **TrueCrypt**
- Retrieves electronic evidence in a matter of minutes from a Windows Desktop Search Database
- Includes a **Portable Version** that runs from a USB thumb drive and recovers passwords without installation on a target PC
- Acquires memory images over FireWire Updated
- Recovers Mac user login passwords from computer memory New!

## Advanced Features

- Instant recovery for many password types
- Acceleration with distributed computing (**Distributed Password Recovery**)
- Multiple-core CPUs and nVidia GPUs acceleration
- **Tableau TACC** hardware acceleration
- 8 different password recovery attacks (and any combination of them) with an easy-to-use setup wizard
- Detailed reports with MD5 hash values



*After losing my password to important encrypted documents, I thought it was the end of the world. Thanks for saving my work, Passware.*

**Conor LaHiff, LaHiff & Company.**

**5**

editions for consumers, small business, professional, corporate, and forensic users.

Starting from **\$49!**

**For additional information, please visit:**

[www.lostpassword.com/kit-forensic.htm](http://www.lostpassword.com/kit-forensic.htm)

**Passware Inc.**

800 West El Camino Real, Suite 180  
Mountain View CA 94040

**Contacts**

Nataly Koukoushina  
media@lostpassword.com  
Phone: +1 (650) 472-3716 x 101

**30**  
DAYS

**MONEY BACK  
GUARANTEE**



# Join Today Free!



Go Premium to support & enjoy the full potential!

New

## Astalavista - The IT News and Security Community

- Forum Posts  SHOW
- Downloads  SHOW
- Events  HIDE
- Official Blog  SHOW
- News  SHOW
- Jobs  SHOW

Astalavista has taken another step into the future.

Stay Up-to-date

With our relaunch we focus even more on the IT & Security world.

Our continuous news stream on the main page gives you all the information you need – 24/7. What do you think about that? Give us a shout on our Astalavista Blog, you find it by clicking on the first news item on our news stream.

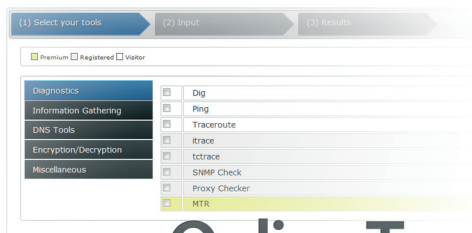
Now  
25%  
OFF!

### Join Today

Use coupon: [hakin9astadiscount](#)

[www.astalavista.com](http://www.astalavista.com)

Go Premium!



### Online Tools

The new **Online Tools** overview page features nearly 50 tools covering typical IT needs, like Whois, Dig, Proxy List or Encryption.

The **Rainbow tables** section lets you hash your plain text in more than forty different types and crack your hashes. The **blacklist checker** runs your domain against the most important black lists and checks if your IP/Domains are flagged as spam.



### Wargames

Wargames by its broad definition is a military drill under real life conditions. It is about testing strategies without the actual combat.

The **“World Gold Reserve”** is where most of the world's gold is stored. The combat in IT is virtual. Here the purpose of a wargame server is to allow you to practice hacker tricks without damaging anything or violating the law. The aim is to find gaps in security and to learn the necessary precautionary actions to prevent this.

Go Premium to support & enjoy the full potential!

IT News and Security Community

# Astalavista.com

No There is no fingerprint, but there is a secret code that gives you a sweet discount: [hakin9astadiscount](#)