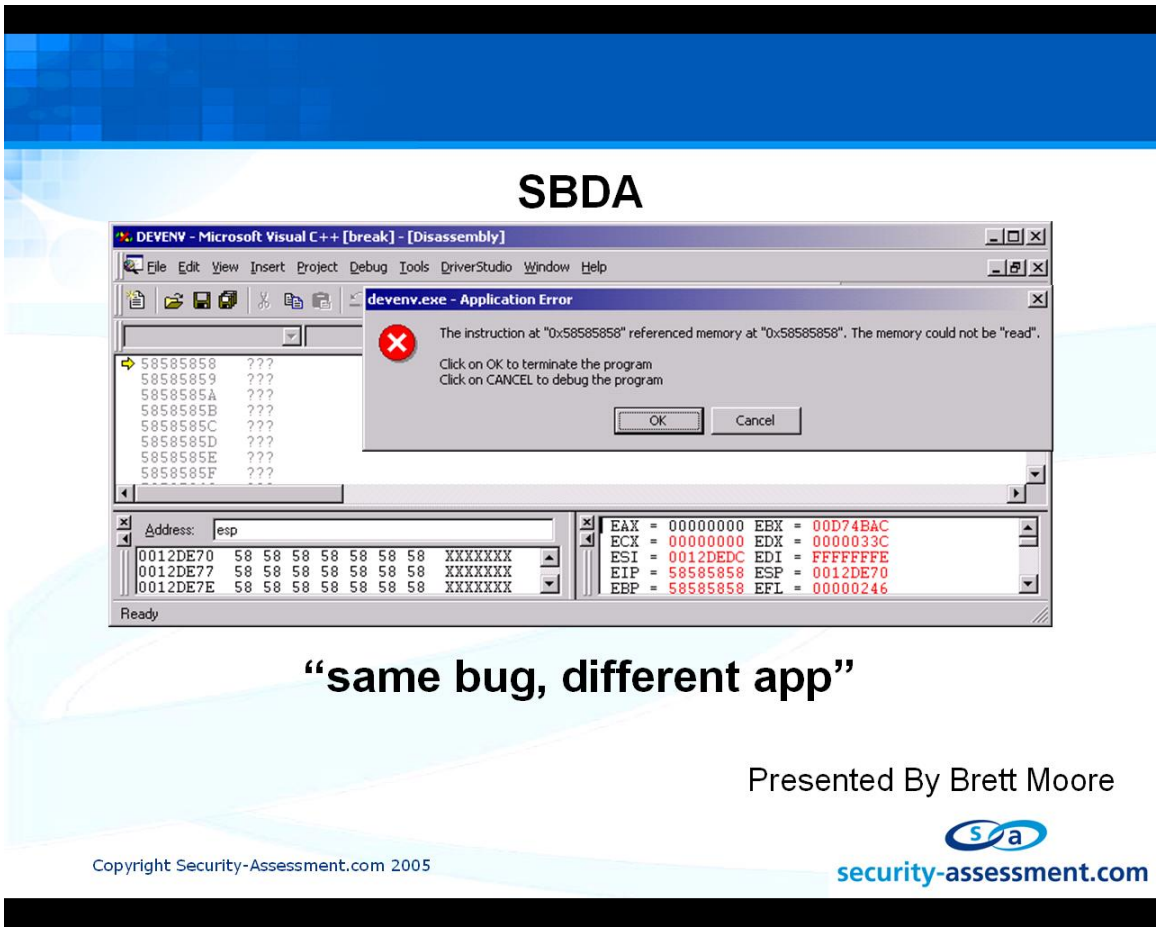
SBDA
Same Bug, Different App



Speech Notes
Ruxcon/BlueHat 2005

Intro

Hi everyone, hope you are all enjoying the day here.

So far the presentations have been fairly outstanding. Its nice to see some overseas speakers making the journey down to this part of the world. And helping make ruxcon what it is…

Please keep any questions till the end and I will be happy to answer them then.

If we don't have time, then feel free to come forward and ask me afterwards.

Slide 2 - SBDA

SBDA – Same bug, different app.

Those that know me often hear me talking about SBDA. It stemmed from my annoyance at the disclosure of yet another vulnerability based on the same vector.

Long filename, long parameter, corrupt file, invalid packet… Same bug, different app. The monotony of it.

To me, SBDA is the theory that an attack vector affecting one application, may have security implications against another application.

As shown on the slide here. An exploit is made of three distinct parts.

The **payload**, is what is sent to an application causing it to do something. This something usually involves the creation of a remote shell. The payload is what is typically referred to as the shellcode.

The **vector**, or attack vector, is the method in which the payload is deployed to the application. This could be a web method such as GET or POST. May be through a FILE, or through a network packet.

The **target** of course, is the application under attack.

The SBDA theory is only interested in the later two parts. The vector and the target.

By modification of the target variable, it is possible to rapidly discover new vulnerabilities exploited through the same attack vector.

The purpose of this presentation is to help you understand this theory, so that you will be able to recognise when opportunities arise to apply it.

Vulnerability discovery is not some magic art that only few can master. Don't get me wrong, some vulnerabilities are incredibly complex, but most follow the SBDA path, and if you haven't realised this yet… you will…

Slide 2 – SBDA – What Its Not

Ok, so we've discussed what SBDA is. I want to just briefly touch on what its not.

When I say 'Same BUG' I'm not referring to the same source code flaw. I'm not interested in whether the vulnerability was caused by a bad strcpy or sprintf. It doesn't bother me if bad data was used in a copy loop, or as an offset into a data structure.

What I'm referring to is the use of the same attack vector against another application.

I'm not talking only about buffer overflows. The SBDA theory applies right across the board. If we applied the theory against web applications, then the top two attack vectors would be Cross Site Scripting and SQL injection.

We're not though. We are talking about binary application vulnerabilities, and the examples in this presentation should show you how this can be applied to any attack vector.

As explained in the previous slide, we are only interested in the vector and target parts of an exploit. The payload is usually vulnerability specific and therefore is by most parts, of no concern to us. Once a vulnerability has been discovered, then we can worry about creating the shellcode and tweaking the payload to our advantage.

Lastly, as is shown there.. These are just my thoughts. And I'm sure I'm not the only one to think like this. In fact later in the presentation I'll show you how another person is publicly disclosing vulnerabilities, most likely discovered using a process similar to what I am talking about.

It's not revolutionary… it's not highly technical… and like a lot of things, in hindsight it's extremely obvious.

But once I realised it, and understood how it could help me in vulnerability research, it's definitely become an indispensable way of approaching discovery of new vulnerabilities.

Slide 4 – Some Historic SBDA

Let me take you back in time for a history lesson. Shown on the slide are some of the past vulnerabilities that help make what I'm talking about so obvious.

Can anyone spot any patterns here?

Lets see now…

Historically there have been a bunch of security issues with how web servers handle requests containing a long name. I've shown a few here.

Historically there have been a bunch of security issues with how web servers handle requests containing a long parameter name. I've shown a couple here..

Patterns anyone?

Historically there have been a bunch of security issues with how web servers handle requests containing a long HTTP header value.

Getting the idea yet?

And of course my favorite.

Historically there have been a bunch of security issues with how web servers handle chunked encoded posts.

This last example is important; as it highlights the fact that SBDA can be applied across multiple operating systems and is platform independent.

Here we have a vulnerability affecting three different totally unrelated webservers running on different platforms.

SBDA – same bug, different app

Slide 5 – SBDA – The Conception

So a long long time ago, in a galaxy far far away… Umm no.. Wrong script.

But it was a while ago… and I really wanted to find a security vulnerability. Not just any vulnerability… it had some specific criteria..

It was going to be remote and it was going to affect IIS.

So I set out on my journey.. Wrote some fuzzers, debugged code, tried so many things I eventually forgot what I had tried, and so ended up trying them again.

What was my target? Everything.. and anything.. I really had no idea what I was doing.. But as far as I know, there was no book titled 'dummies guide to finding security vulnerabilities'.

So I struggled on, and what did I find. Zilch, zip, nada, nothing… Not a god damn thing.

I got to the point where I was starting to believe that "maybe they had all been found"

It wasn't until I was away on holiday that I realised that perhaps I had been going about this all the wrong way. I came to the conclusion that what I needed to do was take a structured and methodical approach to this problem.

And so was born my methodical approach to finding buffer overflows. I realised that it should be possible to methodically attacks IIS components with all known exploit methods. And get results…

But would it work….

Slide 6 – The Original Spreadsheet

So as soon as I got back I set about creating a spreadsheet.. Shown on the slide is the original spreadsheet that I made.. With all its mistakes and other half notes.

Standard spreadsheet stuff.. two axis..

I placed the known attack vectors across the top. These were taken from all the previously disclosed vulnerabilities, and perhaps one or two thrown in of my own imagining.

Down the side went the targets. All the common isapi extensions that handle .asp, .ida requests etc.. Also I placed any accessible files that I found on my installation of IIS. So in went anything from /scripts and /msadc, and the frontpage extension files.

Next step was to go through and mark in those that had already been discovered. These are marked on the spreadsheet as prior vuln.

Then …. The process was pretty straight forward. …. All I had to do was methodically go through and fill in the gaps. So I set off testing the various vectors against each of the targets.

As can be seen I found some…

Three were ones that apparently had been ‘silently’ fixed, as I discovered that my test system was not fully patched, and after installing the latest service pack the vulnerabilities no longer existed.

And three were new, previously undisclosed remote vulnerabilities in IIS components.

So after all the time I had spent, ‘trying things’…this simple but methodical approach had finally led me to discover what I was looking for.

Surprisingly, It took me longer to create the spreadsheet and fill in the previously disclosed vulnerabilities than it did to discover the new ones.

Same Bug, Different App…. Theory proved a success.

10 minutes

Slide 7 – How Can This Be Applied

The slide on the screen is probably missing a point, so I'll mention that first.

As researchers what we should do is create a methodical spreadsheet, or matrix, of all known and previously disclosed vulnerabilities. We should then go through and ensure that all vectors have been tested against all targets.

This is probably almost impossible though, as chances are we won't be able to install and test every application. But we can make a best effort.

Then once we have a base spreadsheet to work from, we need to be alert and watch for new vectors, or new targets that we can place into the spreadsheet. Once a new vector or target is inserted, the methodical testing needs to be done again.

As one of my colleagues has recently realised, this approach can of course be applied to other areas of vulnerability discovery. Namely penetration testing.

As an example, on a recent assignment he charted a list of known attack vectors that affect shell scripts. By using the methodical and structured approach, instead of blindly trying exploitation techniques, he was able to quickly discover and exploit a remote command execution hole.

From a security officers perspective, if you had applied this against your own systems and built a base spreadsheet tracking what you had tested for. Then when a new attack vector was made public, you could quickly and easily test your systems against this new vector.

From a vendor's perspective, any new products or versions of products should be plugged into the matrix, and tested against all known attack vectors.

Now this is incredibly obvious, but if this was happening then there would be no products released that could be exploited by passing a long string or an invalid dimension size, or a large post, or a large filename etc. etc.

From a researcher's perspective, we need to be aware of new attack vectors, or changes to existing ones, that may apply to other applications. For example, recently an old vulnerability has been rehashed. The IIS translate: f vulnerability has come back affecting IIS on FAT32 systems.

The rehash involves the use of Unicode characters in the extension, which appears to bypass the extension protection that IIS has built in. So if Unicode can be used to bypass character based protection here, what other areas may also be affected by this. Since FAT32 webservers are a rarity, nobody is probably

going to look to deeply into this area, but its an example of the type of vector changes to be aware of.

Predicting future vulnerabilities? The slide says yes, but who really knows…

RDP was long overdue a vulnerability… Why?... because it hasn't had one for so long.

Yes it's possible to have a secure application but history has shown that this is rarely the case. The problem is that historically applications were not written with data validation for security purposes.

"Why would the user try and send me a packet that won't do what it's designed to?"

"Why would they try and load a file that won't load like it's meant to?"

"Why would they request a file that can't possibly exist?"

Historically that's how developers would have thought, but in current day that is not the case, and things are definitely improving.

Still though, if you highlight an application that has never had a vulnerability, or very few, its more likely because nobody has had a good hard look at it, or tested all the vectors against it, than the act that it is secure.

Slide 8 – Wild Researchers

All researchers have their own style and approach they take to vulnerability discovery. Listed on the slide shown here, are the most common approaches.

There are those that ‘more or less’ randomly test things. There is no structure to their testing, and they might be trying to test different targets on a regular basis. This is pretty much where I was before I designed my first methodical spreadsheet.

Once a vulnerability is announced it always draws attention to the vulnerable component. If details are not announced then people set about trying to replicate the issue, and re-discover the vulnerability so they can create exploit code.

The nsiislog.dll first patch release is a great example of this when after the patch for the chunked encoding vulnerability was released, HD Moore did some testing and found the large post vulnerability which was still awaiting a patch.

Currently with the reverse engineering tools available, enough binary analysis can be performed to increase the chances of finding any ‘unpatched’ vulnerabilities in a component. This means that if a vendor releases a patch, but has not tested the component for other vulnerabilities, then those security issues may easily be discovered by another researcher.

When a researcher concentrates solely on one target, they may try all known attack vectors, various fuzzing tools and other research methods. If they haven’t charted out the known vulnerabilities though, then they may be looking for vulnerabilities that have already been patched and are therefore wasting valuable research time, or good drinking time.

Researchers who take the methodical approach applying the SBDA theory, should be able to maximize their results for the time spent. In theory a researcher following this method should be able to find all SBDA vulnerabilities in the targets tested.

20 minutes

Slide 9 – The SBDA Advantage

So obviously people find bugs without realising the SDBA theory, or using the methodical approach. So how can SBDA help then?

The first point on this slide is probably the most obvious, and fits with how I found my first chunked encoded post vulns.

“Through mapping out vectors and targets, it allows researchers to easily spot the gaps”.

As I demonstrated on the methodical spreadsheet, the areas of potential exploitation become readily apparent when placed into a spreadsheet. This allows a esearcher to direct their tests against components that are ‘more likely to be vulnerable’

The knowledge gained about attack vectors, while filling in the spreadsheet, means that it becomes easier for a researcher to recognize potential vulnerable situations.. Whether during active research activities or just in day to day computer use. It may also set off ideas of their own, in regards to new attack vectors.

The spreadsheet approach makes it simple to track progress and chart the vectors that have been attempted. This prevents the mistake of trying the same thing every week because you’ve forgotten if you have tried that or not. It also helps to visualize what attacks you still have to test against a particular target, and in some instances allows you to build a picture of areas of a target that you expect to me more susceptible to attack.

“History shows that SBDA works”
If researchers were following the SBDA methods, then all of the historic SBDA vulnerabilities that I showed before, could have been discovered on the day the first was released.

When the long .htr filename vulnerability was announced, a researcher would have tried all other extensions with a long filename, possibly leading to the discovery of the .cfm, .jsp and others.

When the .ida vulnerability was announced, a researcher would have tried all other isapi extensions followed by a long parameter name.

And again, when the first chunked encoded post vuln was announced, all other potential targets would have been tested.

Slide 10 – Some Recent SBDA Trends

Ok so lets get back to the future, to current day.

The SBDA list could go on for ever, and I've never been one for just displaying statistics, so I've listed a few recent ones that are extremely apparent.

The firefox host buffer overflow, was quickly tested against the netscape browser, and most likely others.

Recently a vulnerability was disclosed whereby process explorer could be attacked through a long CompanyName field in an executable. Same Bug as in a long string, but through a new vector. Somebody picked up on this and 'I guess' tried a few things against an application they had installed on their system. They later disclosed that the Killprocess application could be exploited through a long FileDescription field in an executable.

The guys over at SEC-CONSULT recently disclosed that directly loading the jview profiler through the use of the CLSID, would cause a heap overflow in Internet Explorer. Surprise surprise, it was patched, and then multiple other vulnerable components were discovered.

What about all the recent RPC vulnerabilities. Whats the story there.

- Take 1 RPC API that has a string involved as a parameter.
- Capture and dump the packets in motion
- Modify the string or string length values
- Replay packets
- Watch the crash

Any vulnerability based on a known attack vector is a SBDA.
Long filename, long parameter, corrupt packet, corrupt file, user supplied data used in malloc user supplied data used in copy loop, user supplied data used in calculation…

The only difference is the target

Slide 11 – The AV / Archive SBDA

Lately secunia have been releasing a number of advisories about archive and antivirus applications, in relation to how they handle archives.. or more specifically malformed archives.

This list is a great example of SBDA in the real world. Same bug, they all relate to a long string or invalid data. Same vector, they are all exploited through archives. But different target applications.

A good guess would be that secunia have a set of malformed archives, and are testing a wide range of antivirus and archive programs.

As it says, this list is in reverse chronological order. So lets look at the bottom. The guys from ISS, Alex Wheeler and Neel Mehta I think, released a bunch of stuff on AV systems, that could be exploited through archives and other compressed executables.

iDefense picked up on some of these.

And then secunia went to work.

This list shows the results when researchers take an attack vector and apply it against different applications.

Zip and other archive libraries and applications have had numerous vulnerabilities on multiple platforms, and it really makes sense. They use data that is supplied by the file, to perform calculations to uncompress the file. In days gone by programs could do this, because why would somebody try to uncompress an archive that was invalid? This isn't true anymore though as has been shown time and time again.

If you were going to do testing like this, you would want to ensure you had as many attack vectors as possible. What about dir traversal?, What happens if the file is a reserved name? Etc.etc.

And apart from the numerous antivirus systems that could be tested with this particular vector, what other applications may be vulnerable to this type of attack?
Maybe somebody now needs to start looking at mail filters, content filters, spyware detection engines etc..

Slide 12 – Methodical vs Random

As proved by personal experience and as can be seen on the previous slide, a structured testing method can most definitely provide results.

On the same note, random and arbitrary testing can also provide findings.

But by combining the two methods, you can both be assured of finding something standard, and possibly finding something new.

When I say random, I'm not meaning random as in utterly random. I mean random as in not following the methodical approach described before.

Its this trying things outside the norm, that progresses vulnerability research into new areas. I have a lot of respect for those researchers that come up with new attack vectors. Not just new buffer overflows, but perhaps a new way to cause one, or even a new way to exploit it.

The battlefield is changing, with things like SP2 for windows XP, non-executable data areas and also more security conscious programmers.

I mentioned before about how historically programmers didn't expect the user to submit invalid data. They do now, and take precautionary measure around this.

What will happen is that somebody will come up with a new way of doing something that programmers in general didn't expect. And this is where the random testing plays an important part.

Slide 13 – Packet Vs File

A packet and a file are very similar, in that they both are a method of passing data to an application. If an attacker can make a target open a file automatically, then remote exploitation is of course possible as well. But even if this is not the case, a typical user is well warned about opening an .exe extension, and may think that other files are quite safe.

Vulnerability research tends to follow trends. A major interest was shown in remote vulnerabilities against servers, as these are generally the more critical. But now many vulnerabilities have become public caused by malformed files. Interest has also moved into client attacks, where an attacker causes a client to connect to a server that then replies with a malformed packet. All of these avenues are methods of sending exploit data to a target application.

Authough it may seem that file based data is easier to work with when looking for vulnerabilities, once a packet is captured is it not now a file that can be viewed and searched just like any other file.

Slide 14 – Some Common Vector Tests

Listed here are some of the more common attack vectors that should be included in a spreadsheet matrix when applicable.

Long Filename / Path name

Anywhere a filename is used should be tested with a long filename or path, directory traversal and invalid file characters.

Long Parameter

Any parameters or parameter names should be checked

Large Post / Chunked Encoding

Sending of a large amount of data to webservers through all available HTTP methods

String Manipulation

Any obvious text strings should be tested to see what happens when they are replaced with extremely large strings.

Length Value Manipulation

Any obvious user supplied values that appear to be related to size, should be tested

Ok, so how would one go about doing these tests?

Slide 15 – Some Common Test Methods

Fuzzers and fault injectors have been around for years. Some of the more common ones would be Spike from Dave Aitel, and the recently released fileFuzz from iDefense.

All work in the same way. Take a normal set of data, and then start modifyfing the data and sending to the target, in the hope that the target will break.

Manual inspection is probably the quickest and easiest way to spot a potential for a vulnerability. This is where it helps to have knowledge of attack vectors, so they can be applied mentally before trying anything else.

Reviewing the RFCs and data format specifications can also be of use. Once again this may quickly highlight areas that can be tested. Pay special attention to wording such as

"… Must include…", ".. Must not include…", "… specifies the size of….." etc

Making use of disassemblers and debuggers, allows a researcher to get right down to the code level, and find vulnerabilities at the source. There have been a number of tools, released recently that work with disassemblers to automate the discovery of 'potentially vulnerable situations'.

Other automated tools can be used to search for data blocks that have been known to cause vulnerabilities in other applications. A very common file based issue is when the size of the string is stored with the string, which is easily detected with automated tools. Remembering that a captured packet is just a file, means these type of tools can also be uses against network traffic.

Vector automation is a type of fuzzing that allows for the testing of multiple vectors against a single target. This allows the researcher to set up the required parameters and then let it run. After coming back from the pub they can review the findings if any.

And target automation is the sister to vector automation. This type of fuzzing is the use of one attack vector, but applied against a lot of different targets.

Slide 16 – Recognising A Vulnerability

Right… so we know how to look, and we know what to look for.. But how do we know when we have found it?

If you are not prepared when it happens, then you may never know it did.

Always have a debugger on the machine running the application you are testing, and if possible have the debugger attached to the process in question. This means that if an exception happens the debugger 'should' kick in and alert you.

Some exceptions will be handled by the target. If the vulnerable function is inside a try/catch block then the application may silently handle it. So be sure that your debugger is attached to the correct process or processes and that it will break on ALL exceptions. Another useful tool is eFilter which will log ALL exceptions, handled or not.

Its not always possible to attach a debugger tho, especially in the cases where you are testing vectors against numerous 'unknown' applications. So it pays to always be alert. While the event log can sometimes give you an indication of what has happened, an application may close silently and not log anything.

The only reason I noticed the winamp vulnerability was because I came back to the machine and noticed that winamp was no longer playing anything. In fact winamp was no longer loaded.

Be alert.

Don't stop the testing just because you get an error response. Disregards them nd keep going. In fact some error responses should encourage you, as you know that you are causing some reaction from the target.

Also remember that any crash caused by user input, may be exploitable. Some famous words from somebody I can't recall, over a few drinks were along the line of;
"Its only a denial of service because we haven't worked out how to exploit it yet"

If user supplied input is causing the application to behave abnormally, then chances are that if you control the input properly, you will be able to exploit it for arbitrary gain.

But enough talking…. Lets break something…

Slide 17 – SBDA Theory In Practice

Ok.. now the following slides attempt to describe some of the vulnerabilities that I have discovered, and the process used to do so.

They are not new bugs, most have been patched, and all are based around the SBDA theory that I have been talking about.

As the slide says though, pay attention and you could take away information that you will find useful when doing your own vulnerability research.

Slide 18 - The Long Filename SBDA

It's October 12, 2004 and Microsoft release a patch for a buffer overflow in the group converter tool. The overflow was caused when the tool attempted to load a .grp file with an extremely long filename.

That was it for me. I'd basically had enough and couldn't comprehend how a company like Microsoft still had vulnerabilities of this nature in their software.

So I wrote FileNameSizer. This tool creates files with the maximum filename length possible. Each of these files had a different file extension ranging from aaa through zzz.

FileNameSizer tool would then attempt to load each file through its default application. Most of these files had no handler and therefore did nothing.

But as can be seen on the slide, on my windows 2000 installation, I discovered four target applications vulnerable to buffer overflows through the long filename attack vector.

Same Bug, Different App

Slide 19 – The Long Value SBDA

Many vulnerabilities are the result of invalid data that is then used in the allocation of buffers, the copying of data, or offsets into data blocks.

A common storage method is to store the length of the data in the word or dword before the actual data. This can lead to various exploitable scenarios.

As can be seen here, an application may allocate a buffer of the size given in the data, and then copy the data until a null byte is encountered. Therefore overflowing the buffer if it has been provided a smaller length value.

It may read in the value, and then add a byte or two to accommodate for null terminators, and then read in the number of bytes worth of data. If however the malicious data provides a value that causes an integer wrap around, then a smaller buffer is allocated than there is data to copy. Therefore overflowing the buffer.

The application buffer may be allocated a specific size, but then the application reads in the length value, and then reads that amount of bytes from the supplied data. Therefore overflowing the buffer.

These are just some examples and relate mainly to text strings within data blocks. Other values could be used in calculations or offsets that may also lead to exploitable conditions.

FileLengther was a tool that a colleague and I created to search through files looking for text strings that had their length stored in the data bytes before the string.

Some examples of these vulnerabilities.

As can be seen here in this data taken from an excel spreadsheet file, we can clearly see ‘SHEET2’ and the word before it has a value of 6. 6 is the length of the text SHEET2.

The circled value FFFF highlights the start of the exploit. That FFFF originally read 6 followed by ‘SHEET3’. We replaced the length value with FFFF, followed by a large amount of text, leading to a buffer overflow.

The next example is from the htmlhelp .chm vulnerability, and was exactly the same bug except for the value been stored in a DWORD.

Same Bug, Different App

Slide 20 – File Fuzzing

We mentioned before about fuzzing as a method used to test for vulnerabilities. File fuzzing is another quick and easy way to locate new bugs. File fuzzers should do things such as;

- Extend an existing string by overwriting the following data
- Insert to an existing string so as to push out the following data
- Modify each byte/word/dword to particular values
- Then try and load the file.

Attempting this against almost ANY file will cause exceptions. In most cases it will cause so many that it takes longer to determine if the finding is exploitable or not then it did to do the fuzzing.

So we have our own file fuzzing tool that was created as part of our file based testing research.

Let's have a quick look at a fuzz.

Slide 21 – The Long Import SBDA

January 2005 and a vulnerability is disclosed affecting IDA pro. The attack vector is through a long import name in the PE header.

Suddenly researchers are alerted to a bunch of new targets. Debuggers and dissasemblers are now targets for exploitation and a bunch of advisories follow as are listed on the slide there.

A true SBDA follower would now create a spreadsheet, place the vectors and targets on it, Fill in the already known, and try for the gaps.

Due to the limited vulnerable audience though, we only tried a few things with what was on one of our machines. Still though, this led to the discovery that the same vector affected depends.exe, which is the dll viewer installed with visual studio.

Same Bug, Different App

Slide 22 – The Long Text SBDA

An unexpected long string, is one of the most common vulnerabilities discovered. This type of bug has affected numerous applications across all platforms, and is what creates the typical buffer overflow.

Three step process

1) find a file / packet that has string inside it
2) Change this string to a really long string
3) Load the file, send the packet

Some of our findings are shown here.

The .cbo file was a two line file. By inserting a long string for the username a buffer overflow was caused in the Microsoft Interactive Trainer.

A visual studio solution file has a string for the project name. Setting this to a long string and loading it causes a buffer overflow in visual studio.

The hyperterminal session file stores the connection port. Inserting a long string here and having hyperterminal open it will caused a buffer overflow in hyperterminal.

A .job file stores the path and filename of the process to load. Setting this to a long string caused a buffer overflow in the task scheduler.

All of these were discovered through simply looking at the file in a text/hex editor and making the changes.

Same Bug, Different App

Slide 23 – The URL SBDA

Right.. so there have been a few of these vulnerabilities on windows, and on other platforms as well. Recently a bug came out for FireFox and Thunderbird due to their handling of parameters passed through a URL.

We took notice back when the mailto: bug was released. Interest, because to us this was a new attack vector to watch. Unfortunately due to work load we didn't follow it up until after the release of the notes: handling bug.

Then SBDA kicked in and we set about searching the registry for other URL Protocol entries. Once an entry was found we checked out what parameters could be passed to the underlying program, and if they were exploitable.

The hyperterminal one tied into around the same time we discovered the .ht session file overflow. And we used the URL handling bug to force the target to load our session file.

Another telnet program we tried had a similar issue, whereby a configuration file could be specifed, leading to script execution on a target machine.

There are sure to be plenty more of these, affecting all different types of applications.

One thing to note is the # character in the URL on the slide. This was required to prevent the modification of the forward slash into a backslash, and may be of use when trying to discover these types of vulnerabilities in other applications.

Slide 24 – The CLSID SBDA

So this one should still be fresh in everyones mind. July 2005 and a bug report is released explaining how the loading of the javapry.dll file within IE can cause a heap overflow.

A new vector and a bunch of new targets. Perfect for SBDA.

And as expected the month later a patch was released to fix the problem affecting more than 20 different components.

Still though, a quick test shows there are still problems in there somewhere.

On both our win2k and xp machines, we pulled out all the CLSID's from the registry. A quick bit of search replace to add the HTML object loading code.

Load it in Internet explorer, watch and weep.

Slide 25 – SBDA To The Test

Well you've sat there and listened to me ramble on about SBDA.. So lets put It to the test.

Due to corporate regulations I can't disclose any real 0day vulns here. But I do have a good example to show, so bear with me and we will see if SBDA works.

DO THE DEMO

So what that shows is how easy it is to discover a SBDA vulnerability once you know what it is you are looking for. Also it showed that just because a bug doesn't initially look exploitable, a little bit of data manipulation may allow you to direct execution down a path that puts you in control.

It also highlights how the XP service pack 2 heap protection does not prevent the exploitation of all heap overflows.

Slide 26 – Weakness With Common Tests

Fuzzing is more or less guesswork. It is the automated attempt of sending malformed data at an application, and will not find all vulnerabilities. One thing to note though, is to ensure that the requests you are sending were valid in the first place. Otherwise the fuzzer might be running and not returning results because the application is not accepting the format that you are sending.

Of cause with target automation, only the applications installed on the test system can be tested. There will always be cases were another person will find bugs using the same tests you have used, because they have a different application installed.

Lets throw some more into the equation. If we take our filenamesizer tool, that created filenames of the maximum length and tried them. What if a vulnerable application would reject files with a name over 200 bytes in length, but elsewhere it had allocated a buffer of only 100 bytes for the name. FileNameSizer would have missed this bug. A more complete test would be to create files of all lengths for all extensions. It would take longer to run, but there is a chance that it would find more bugs.

Different byte values affect applications differently. Sending a long string of X's might not cause a vulnerability to expose itself. But sending a long string of A's might. This could be due to the application reading the data value from the heap and taking a different execution path, where only one path leads to the vulnerable code.

On that note, what if the application required A's in part of the buffer, and X's in another part. What if the application only accepted data that had at least one backslash in it, or if it required a string that had two path identifiers.

The possibilities are endless, but these sort of bugs may be already being missed by current day fuzzers.

Slide 27 – 2nd Generation Vulnerabilities

Recently a colleague of mine discovered one of these in an unnamed service. He had captured a packet and then gone about fuzzing the data of the packet and watching for results.

Authough we eventually discovered that it was a non exploitable situation, the vulnerability required that a byte of the packet was set to 0, and then a different word could be used to control the offset into the buffer during a copy operation.

I have yet to see a fuzzer capable of testing for these types of situation, but have no doubt that they will be created eventually. The only difference is really a time based thing.

How long would it take to iterate every value through every byte of a 200 byte packet?

This may be beyond the extent of fuzzing tools but with the advancement of static binary analysis tools, these types of vulnerabilities will eventually be discovered as well.

Slide 28 – Non SBDA Vectors

Non SBDA vectors are the ones to watch for, especially if they can be applied against different targets.

They may then become standard SBDA attack vectors and lead the way to the discovery of bugs in different applications.

To me, these are the real interesting ones. It raises the bar on research, sometimes opening the door to a whole new avenue of attacks and vulnerabilities.

It also distinguishes those researchers who are actively thinking outside the square, and trying things that perhaps nobody has tried before.

New and previously unreleased information is what pushes the boundaries of security research, and if people always stuck to the same bug, different app approach to vulnerabilities then this type of research would never occur.

Slide 29 – Wrap Up

So that's about that then. I've explained the SBDA theory and how it can help in the discovery of new vulnerabilities.

I've shown examples of a bunch of different vulnerabilities, all discovered using this approach, and displayed how easy it can be to find new bugs in different applications.

I've shown how SBDA crosses the platform boundaries, and how as researchers we should be testing these vectors against targets on different platforms.

And hopefully I've passed on to you some interesting information that will allow you leave here and apply this knowledge in your own vulnerability research.

Thanks for listening, are there any questions?