How to blackbox test almost anything

Aviram Jenik, CEO

beyond security

My goal here today

Recipe for finding unknown security holes

My goal here today

Recipe for finding unknown security holes

On every platform

My goal here today

Recipe for finding unknown security holes

On every platform

For every programming lang.

My goal here today

Recipe for finding unknown security holes

On every platform

For every programming lang.

Any product type

Blackbox Testing

Is blackbox testing really that powerful?

This week's major security holes on SecuriTeam.com:

- **F5 Denial of Service**
- **Mozilla Firefox disclosure vulnerability**
- **Cisco TCP connection DoS**
- **Adobe Flash Player code execution**

Datastor Kernel Denial Of Service Vulnerability | 0 Comments — 16 Feb. 2016

The datastor kernel module in F5 BIG-IP Analytics, APM, ASM, Link Controller, and LTM 11.1.0 before 12.0.0, BIG-IP AAM 11.4.0 before 12.0.0, BIG-IP AFM, PEM 11.3.0 before 12.0.0, BIG-IP Edge Gateway, WebAccelerator, and WOM 11.1.0 through 11.3.0, BIG-IP GTM 11.1.0 through 11.6.0, BIG-IP PSM 11.1.0 through 11.4.1, BIG-IQ Cloud and Security 4.0.0 through 4.5.0, BIG-IQ Device 4.2.0 through 4.5.0, BIG-IQ ADC 4.5.0, and Enterprise Manager 3.0.0 through 3.1.1 allows remote authenticated users to cause a denial of service or gain privileges by leveraging permission to upload and execute code.. More ›››

Mozilla Firefox URL Information Disclouser Vulnerabilities | 0 Comments — 15 Feb. 2016

The Search feature in Mozilla Firefox before 42.0 on Android through 4.4 supports search-engine URL registration through an intent and can access this URL in a privileged context in conjunction with the crash reporter, which allows attackers to read log files and visit file: URLs of HTML documents via a crafted application.. More ›››

Cisco Virtual Topology System TCP Connection Functionality Denial Of Service Vulnerability | 0 Comments — 14 Feb. 2016

Cisco Virtual Topology System (VTS) 2.0(0) and 2.0(1) allows remote attackers to cause a denial of service (CPU and memory consumption, and TCP port outage) via a flood of crafted TCP packets. More ›››

Adobe Flash Player And AIR Arbitrary Code Execution Vulnerabilities | 0 Comments — 13 Feb. 2016

Use-after-free vulnerability in Adobe Flash Player before 18.0.0.268 and 19.x and 20.x before 20.0.0.228 on Windows and OS X and before 11.2.202.554 on Linux, Adobe AIR before 20.0.0.204, Adobe AIR SDK before 20.0.0.204, and Adobe AIR SDK & Compiler before 20.0.0.204 allows attackers to execute arbitrary code. More ›››

What do they have in common?

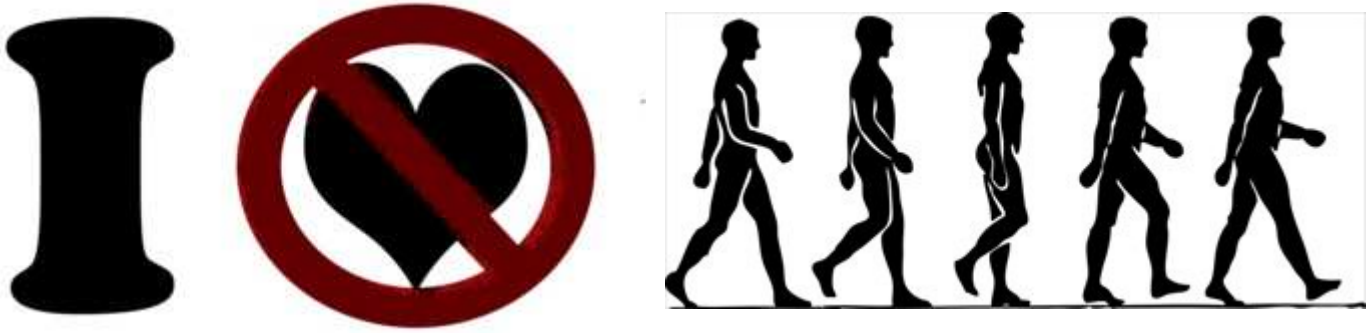
- 4 different products
- Different attack vectors
- All critical vulnerabilities
- All require patch
- Unpatched systems will be extremely vulnerable
- All could have been discovered during development
- None requires special expertise to exploit (hence, relatively straightforward to discover)
- (probably) found via fuzzing – e.g. blackbox testing

Blackbox testing in the real world

This was my week:

Monday	Tuesday	Wednesday
Bluetooth attack (unpaired) on a car hands-free system	Malformed WAV file vulnerability on a car entertainment system	CAN BUS (OBD-II) request caused full lock-up of the car (required towing)

Something about me



My goal here today

Finding those vulnerabilities automatically (machine-style)

About Beyond Security

We specialize in vulnerabilities

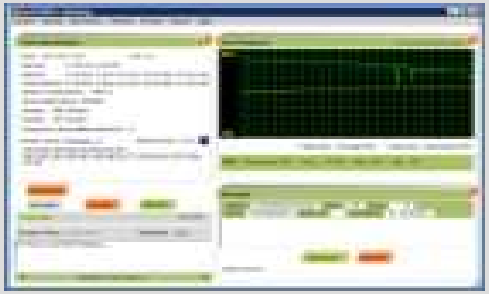
About Beyond Security

We specialize in vulnerabilities

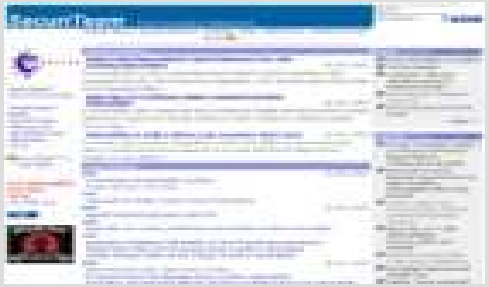
and develop tools to find them

Our Technology

AVDS



beSTORM



SecuriTeam.com

Our Technology - AVDS

AVDS

Automated Vulnerability Detection System ("VA")

Know that your network is **Safe**

AVDS

beSTORM

SecuriTeam.com

Known Vulnerabilities

AVDS

(Automated Vulnerability Detection System)

- Everything that talks 'IP".
- Agent-less, providing a real 'hacker' view
- Scalable from 64 to many hundreds of thousands of systems
- Very powerful management and automation tools

AVDS

beSTORM

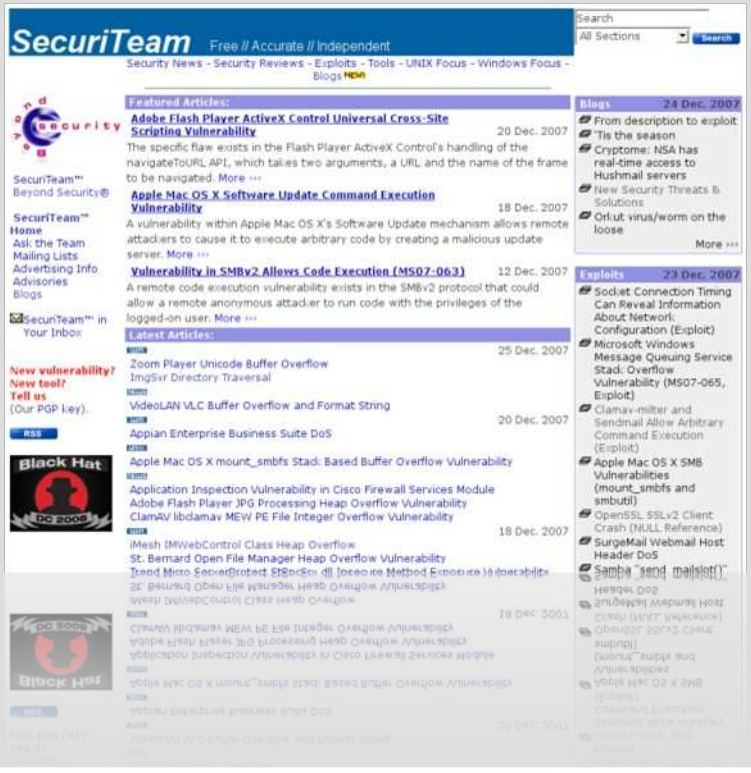
SecuriTeam.com

Our Vision with AVDS

- Scanning 1-4/year is just not enough any more
- Vulnerability Management is an active process
- scans must be done on a regular basis:
  - New vulnerabilities are discovered every day
  - The network is dynamic (new ports, services, hosts)
- Needs to be a dedicated, robust platform (similar to Firewall/Proxy/IPS)

The objective of Vulnerability Management is to KNOW, at any given time, what the risks are in your infrastructure so that they can be managed.

Our Technology - SecuriTeam



SecuriTeam.com

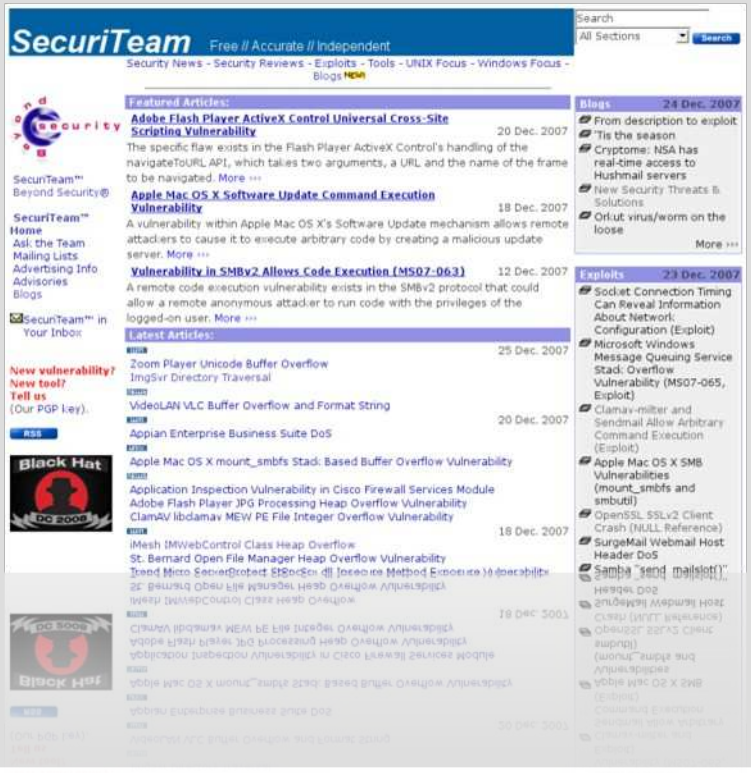
Security portal / Knowledge Source

SecuriTeam.com

AVDS

beSTORM

Information



SecuriTeam.com

Security portal / information source

- All information/mailing lists are free
- Global gathering place for IT Security Professionals and Hackers
- One of the leading portals worldwide on vulnerabilities and exploits

SecuriTeam.com

AVDS

beSTORM

Among our customers

How are cyber attacks done?

"The reality is that the most important issues are the vulnerabilities and the techniques used to exploit them, not the country that appears to be the source of the attack"

- Gartner

How was the recent cyber-incident done? Most likely by a vulnerability that is easy to uncover and patch

The theory behind Blackbox testing

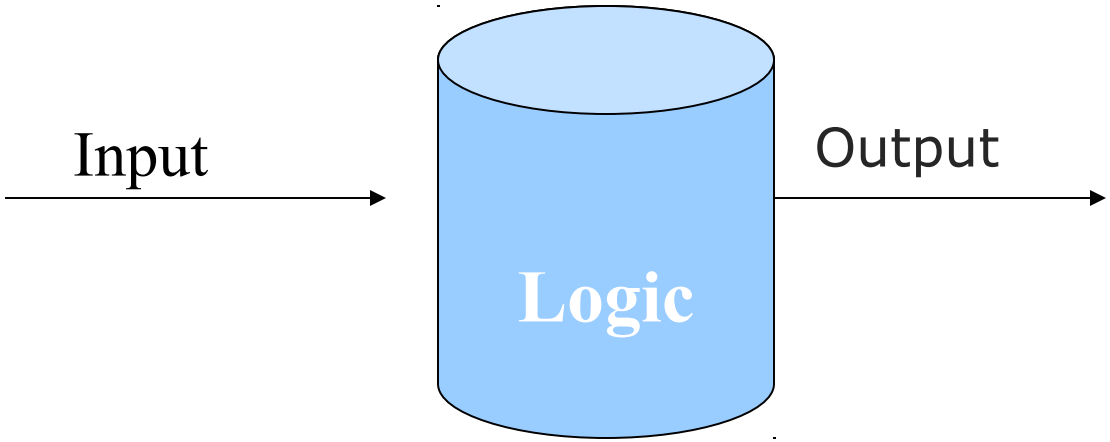
beyond security

The most secure system

Is completely useless

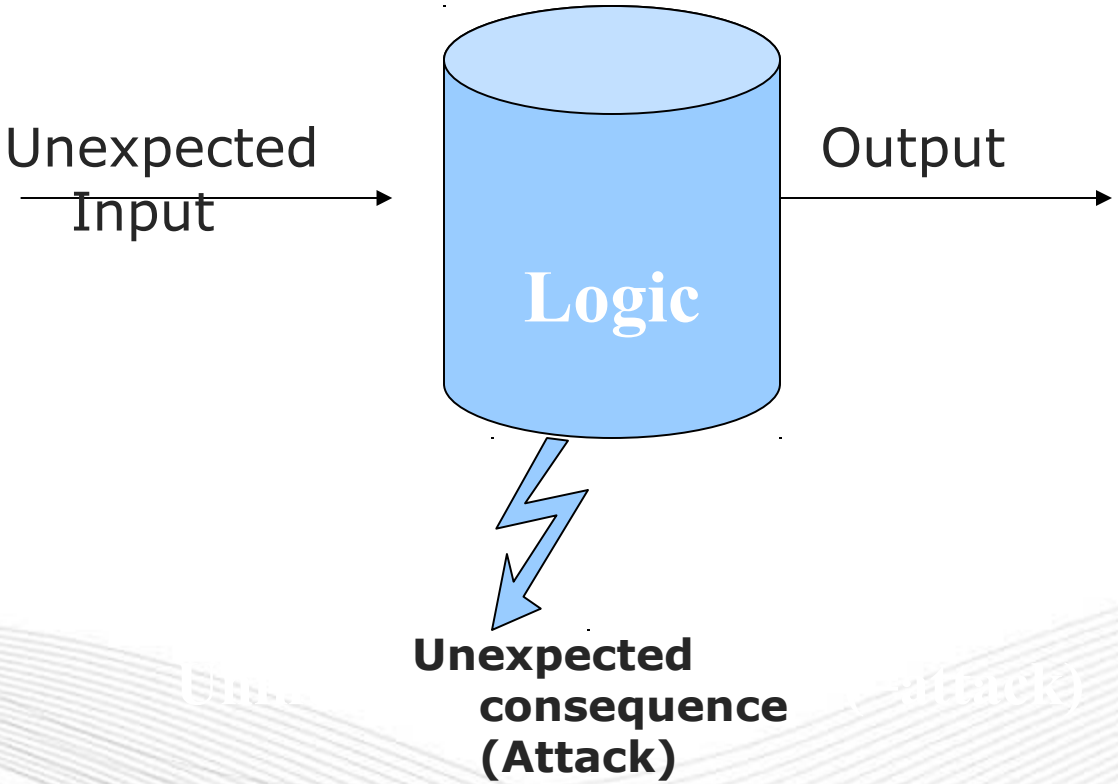
Inputs are the problem

For the programmer

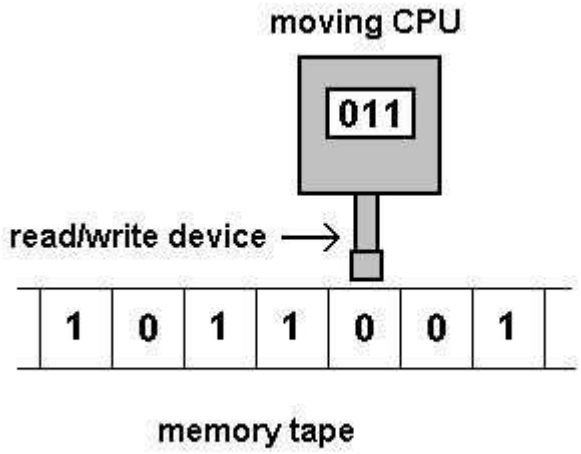


Inputs are the problem

For the attacker



Turing machine



The basic software model

What is blackbox testing?

- Testing by attacking the inputs and observing output/behavior
- Does not use the source code
- Does not assume knowledge about the system

Doesn't QA solve this?

QA: Testing if a good input => good result

Fuzzing: Testing if a malformed input => good result for the attacker!

What is blackbox testing?

- Testing by attacking the inputs and observing output/behavior
- Does not use the source code
- Does not assume knowledge about the system

The system is a black box

What is blackbox testing?

- Testing by attacking the inputs and observing output/behavior
- Does not use the source code
- Does not assume knowledge about the system

The system is a black box

This is how almost all security holes are discovered today

Exhaustive blackbox testing

The theory: Generate all possible combinations

Exhaustive blackbox testing

000...000

000...001

000...010

.

.

.

.

111...111

Exhaustive blackbox testing

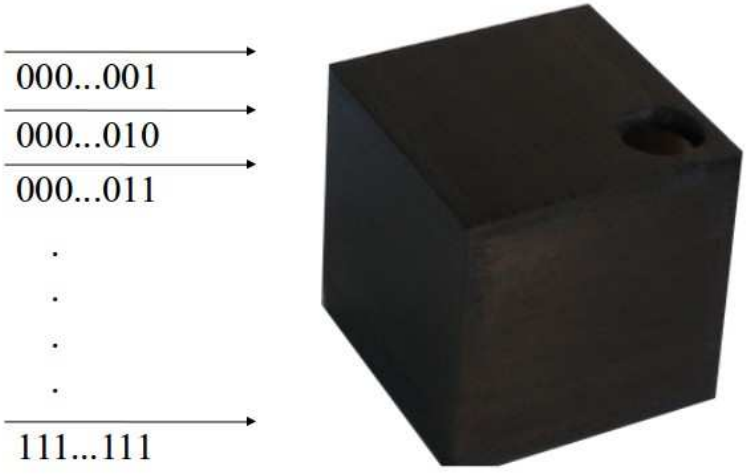
All possible inputs ==> All possible outputs

Exhaustive blackbox testing

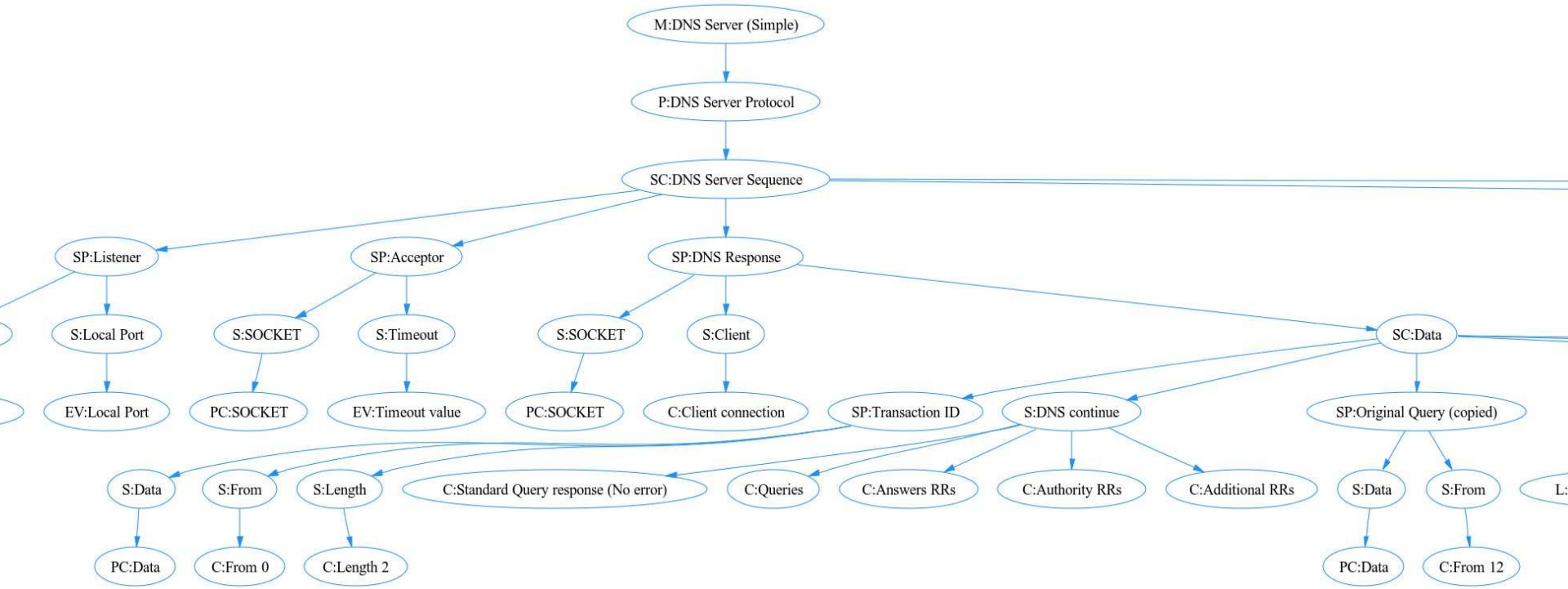
All possible outputs ==> All possible security vulnerabilities will be triggered

Exhaustive blackbox testing

The drawback: 1KB request = $2^{1000}=10^{300}$ combinations



Shortcut #1: Protocol coverage

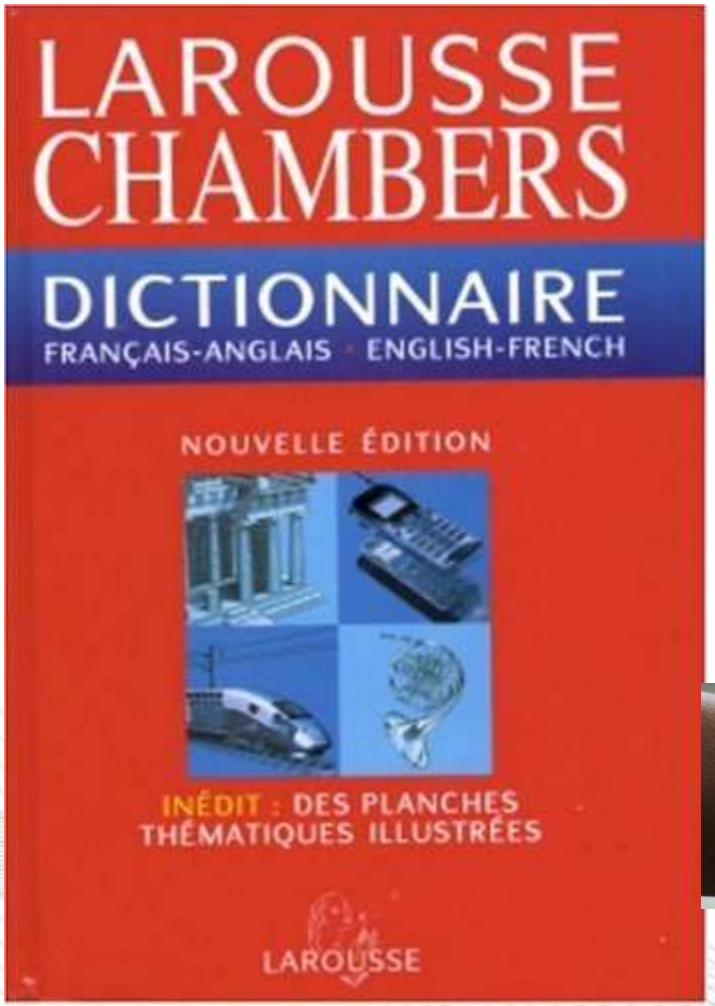


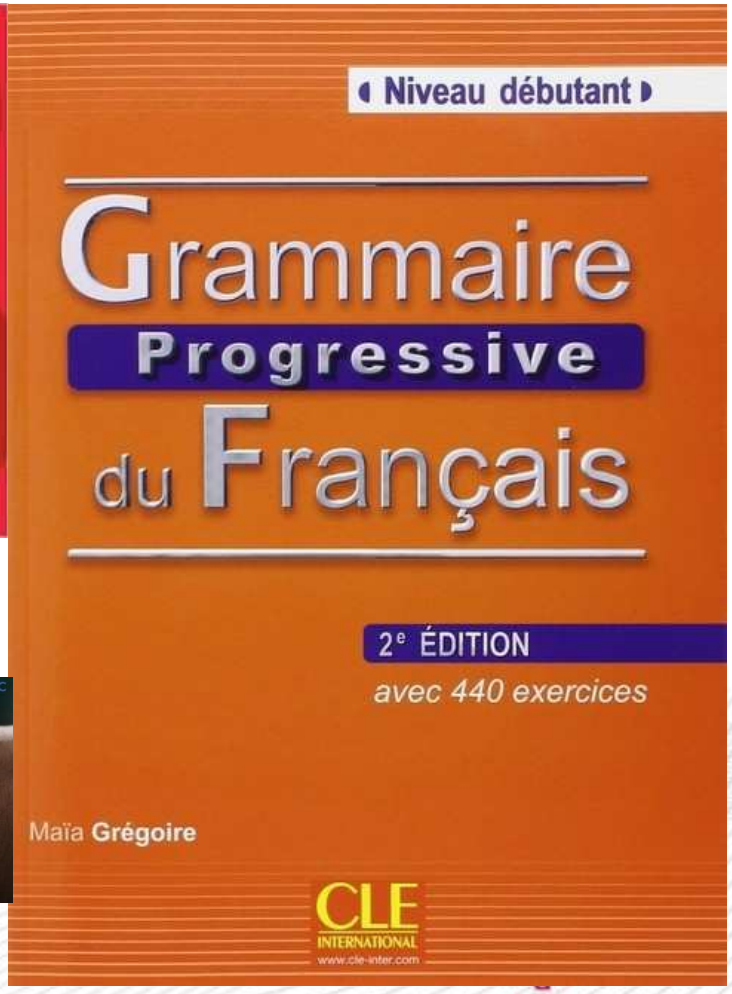
Example

Blackbox Testing French people

Example

Blackbox Testing French people





Shortcut #1: Protocol coverage

Step 1: Generate all possible valid protocol requests (by crawling through the BNF description of the protocol)

Shortcut #1: Protocol coverage

Step 1: Generate all possible valid protocol requests (by crawling through the BNF description of the protocol)

==> Guaranteed to cover the entire protocol

Shortcut #1: Protocol coverage

Step 1: Generate all possible valid protocol requests (by crawling through the BNF description of the protocol)

==> Guaranteed to cover the entire protocol

Step 2: "Fuzz" (=attack) each field in each combination

Shortcut #1: Protocol coverage

Step 1: Generate all possible valid protocol requests (by crawling through the BNF description of the protocol)

==> Guaranteed to cover the entire protocol

Step 2: "Fuzz" (=attack) each field in each combination

Attacks:

- **Buffer overflow: AAAA...AA**
- **Format string: %n**
- **Null character: 0x00**
- **XML attacks: < and >**
- **Space**
- **Various encodings**

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

USER AAAAAAA...A (n times)

What is 'n'?

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

USER AAAAAAA...A (n times)

What is 'n'?

Naive Solution: Test everything.

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

USER AAAAAAA...A (n times)

What is 'n'?

Naive Solution: Test everything.

USER A

USER AA

USER AAA

…

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

USER AAAAAAA...A (n times)

What is 'n'?

Naive Solution: Test everything.

USER A

USER AA

USER AAA

…

"n" combinations

Shortcut #2: Scaling quickly

Problem: Buffer overflow testing requires many combinations:

USER AAAAAAA...A (n times)

What is 'n'?

Smarter option: Scale quickly 2^n

USER A (2^0)

USER AA (2^1)

USER AAAA (2^2)

USER AAAAAAAA (2^3)

$\log_2(n)$ combinations

Shortcut #2: Scaling quickly

Smarter solution can test buffers from 1 byte to 64KB in 16 steps (naïve method takes 65,535 steps)

Still covers small buffer sizes (2, 4, 8)

Covers medium buffer sizes (1024, 2048)

And covers large buffers (32,000 and 64,000)

Smart Fuzzing

Conclusion: It's possible to do an 'exhaustive' testing while taking a few shortcuts to reduce the combination count without reducing quality

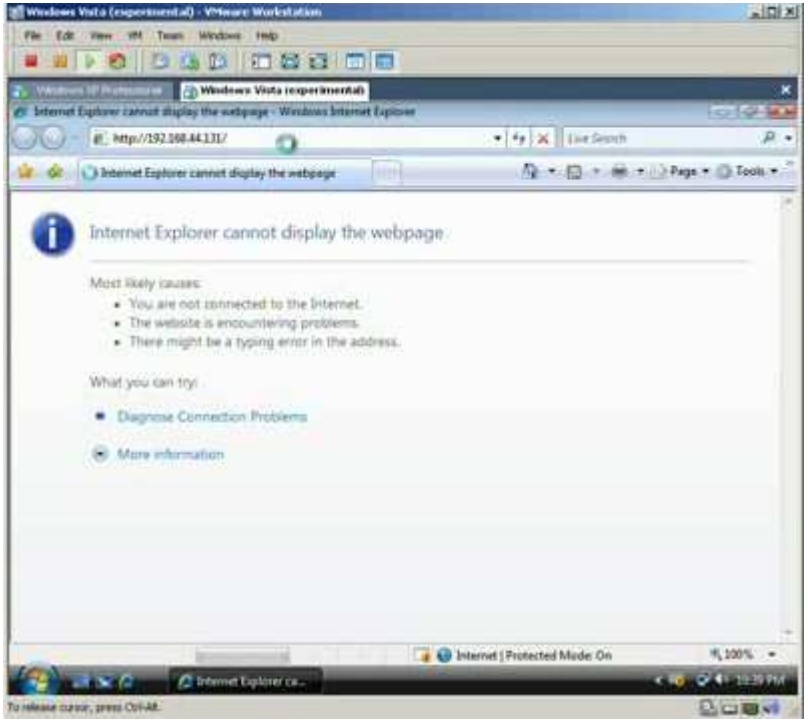
How to blackbox almost everything

Step 1: map all your inputs – in production

- **File inputs**
- **Network**
  - IP
  - Wireless?
  - RFID?
- **Library calls**
- **Command line parameters**

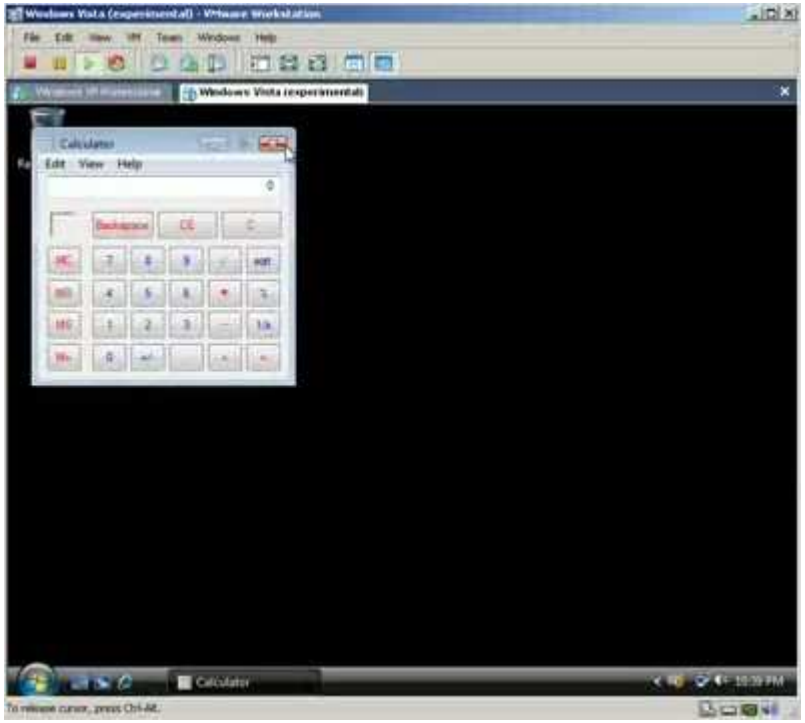
Why file input can be especially dangerous

"preview" - ANI attack



Why file input can be especially dangerous

"preview" - ANI attack



Who determines risk?

Not you!

Who determines risk?

Attackers attack what's easy and not where you ask them

How to blackbox almost everything

- **Step 2: determine your "protocols"**

Protocols

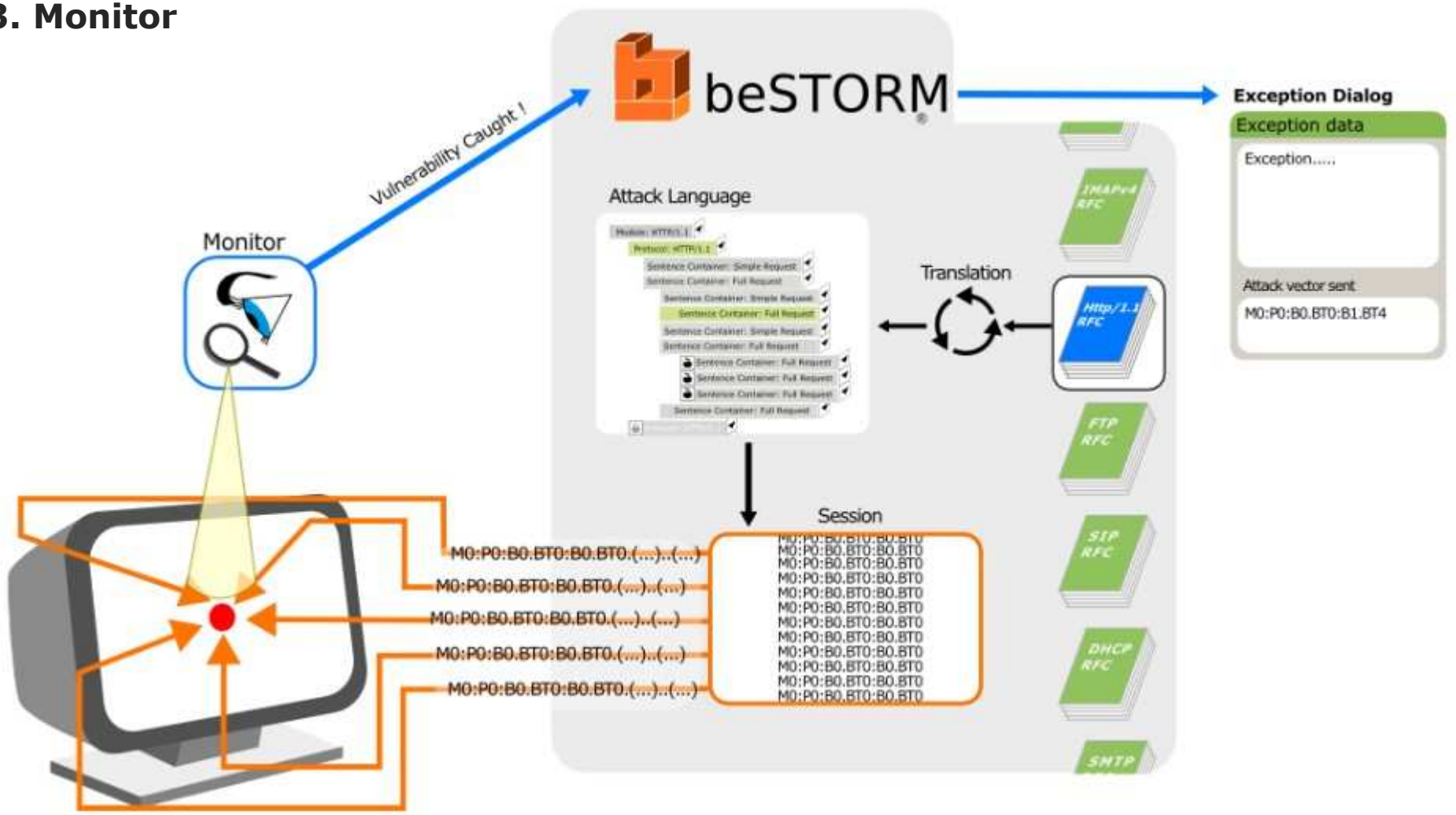
- **Network: your RFC (or spec-based) protocol**
- **File: accepted file formats**
- **Library: DLL/ActiveX Interface**

How to blackbox almost everything

- **Step 3: Start testing**

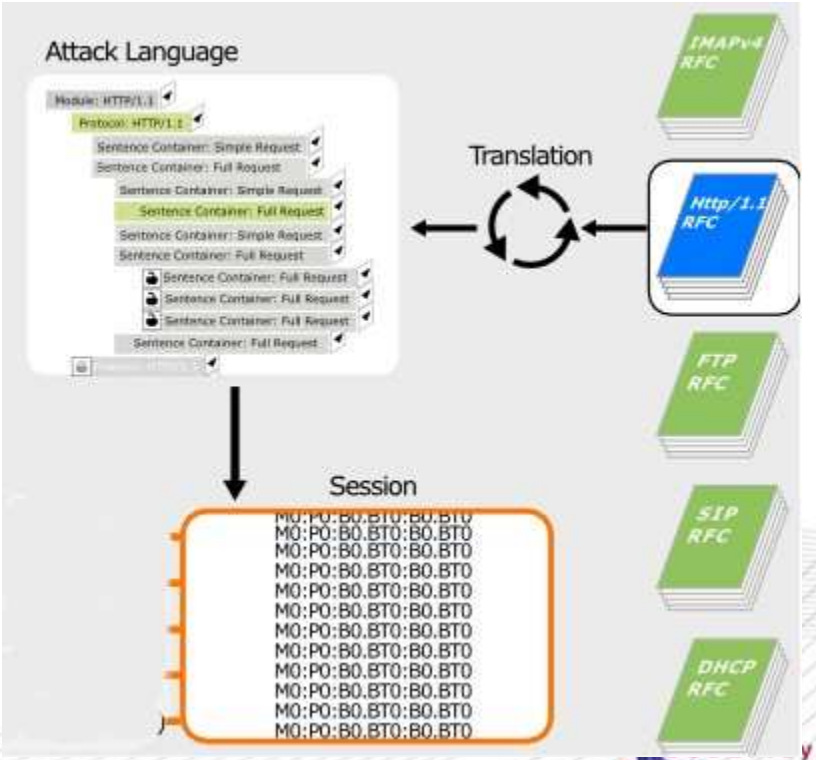
Ingredients

1. Test Module description
2. Generator
3. Monitor



Test Module

- **Something that can describe “many” “different” sessions (=attacks)**
- **Protocol coverage is key**



Example: beSTORM BSP file format

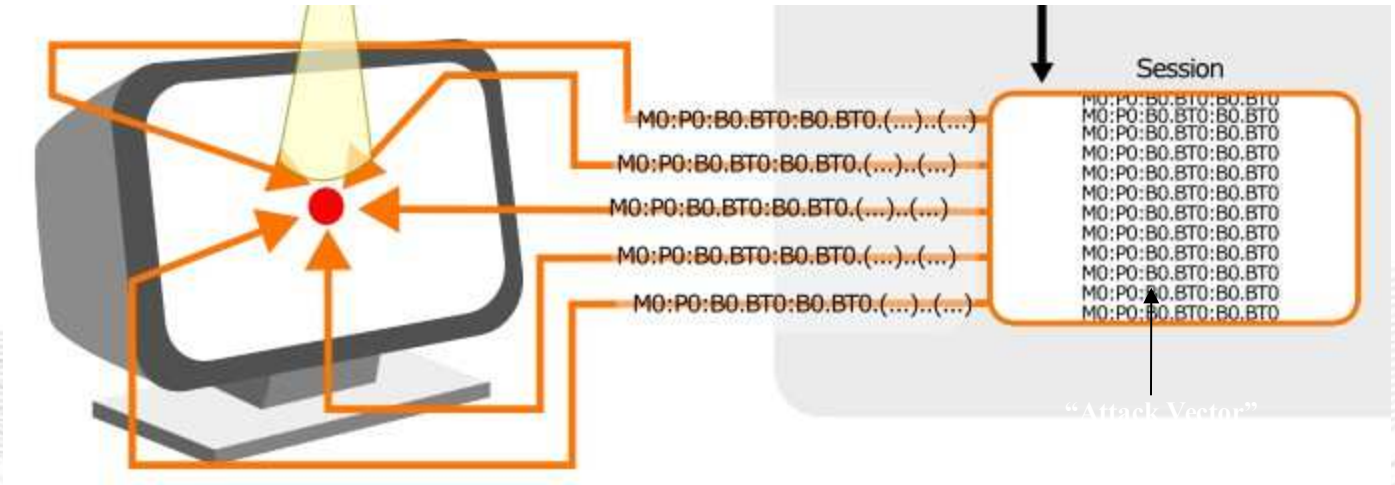
```
<SC Name="ICAP Request">
 <SE Name="ICAP Method">
  <S Name="ICAP Method Enumerating">
   <E Name="ICAP Methods">
    <C Name="REQMOD Method" ASCIIValue="REQMOD" />
    <C Name="OPTIONS Method" ASCIIValue="OPTIONS" />
   </E>
  </S>
  <S Name="ICAP Method Overflow">
   <B Name="ICAP Method Overflowing" ASCIIValue="RESPMOD" />
  </S>
 </SE>
 <S Name="Request Line">
  <C Name="Space" ASCIIValue=" " />
  <B Name="icap Prefix" ASCIIValue="icap" />
  <C Name="ColonSlashes" ASCIIValue="://" />
  <B Name="Address" ASCIIValue="10.50.10.71" />
 </S>
</SC>
```

Example: beSTORM BSP for file fuzzing

```
<M Name="TGA" >
        <P Name="TGA Protocol" >
          <SP Name="Writer"  Library="File Utils.dll"
   Procedure="Write">
              <S Name="Path" >  <VB Name="Whatever" Description="Path to store
   files" NoDefaultTypes="1" ASCIIValue="c:\\temp" />  </S>
              <S Name="Directory Splitter" >
                 <VB Name="Whatever" Description="Directory Splitter
   size" NoDefaultTypes="1" ASCIIValue="2" />
              </S>
              <S Name="Extension" >
                 <VB Name="Whatever" Description="Extension"
   NoDefaultTypes="1" ASCIIValue="tga" />
              </S>
              <SC Name="Data" >
                 <S Name="Color-mapped images" >
                    <L Name="Identsize"  ConditionedName="Image
   Identification Field" Size="1" />
                    <B Name="Colour Map Type"  Default="0x00"
   MaxBytes="1" />
                    <B Name="Image Type Code"  Default="0x02"
   MaxBytes="1" />
                    <B Name="Color Map Origin"  Default="0x00,0x00"
```

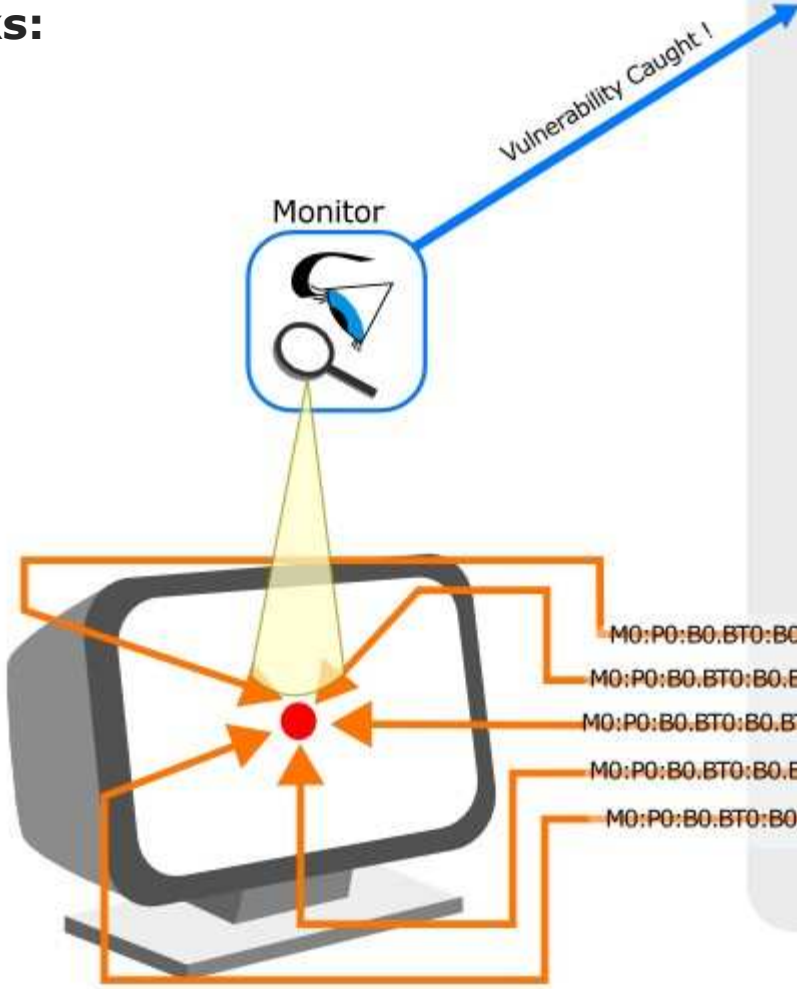
Generator

- **Something that can take the module description and send it to the program:**
  - Over the network
  - By creating a file
  - By invoking a DLL function



Monitor

- **Possibly the most important component**
- **So you're generating millions of attacks: but how do you know you succeeded?**



Monitoring

Monitor for:

- **Memory exceptions (“first chance exceptions”)**
- **Program stops responding**
- **Errors in Logs (via regex)**

- **Connect the monitor with the generator to correlate**

Easy to use and extend

- **Windbg**
- **gdb**

Key factors

- **Automation**
- **Re-creating the attacks**
- **Ensuring protocol coverage (not code coverage!)**

Report Sample

Thank you!

Questions?

aviram@beyondsecurity.com